Speeding Up Hardware Verification by Automated Data Path Scaling

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Peer Johannsen

Kiel,
im August 2002

1. Gutachter:	Prof. Dr. Th. Wilke
2. Gutachter:	Prof. Dr. W.-P. de Roever
Datum der mündlichen Prüfung:	18. Dezember 2002

Acknowledgments

Ask, and it will be given you,
seek, and you will find,
knock, and it will be opened to you.

For every one who asks receives,
and he who seeks finds,
and to him who knocks it will be opened.

– Matthew, Chapter 7:7 –

Per Aspera ad Astra ...

– A rough road leads to the stars –

The completion of this work brings to a close a long journey. Many have helped me along the way, but rather than thanking a great number of people, I want to thank a number of great people. People who have assisted me and without whom I could not have succeeded. This work would not have been possible without their help and support.

First of all, I want to thank my supervisors Thomas Wilke and Willem-Paul de Roever at the University of Kiel for reviewing my thesis and for agreeing to a long-distance collaboration and supervision. Their careful advice has considerably influenced my work. Furthermore, I want to thank Margrit Krause and Erich Valkema for their help and for the very pleasant atmosphere at the chair during all my visits to Kiel. In addition, I want to thank Wolfgang Thomas at the RWTH Aachen, whose teaching and guidance has been a constant source of inspiration and help over the last years.

Secondly, I want to thank Siemens Corporate Technology and Infineon Technologies for giving me the opportunity to do the research that resulted in this thesis at their research centers in Munich, Germany, and for providing a highly inspiring research environment. Working between the two worlds of academia and industry and being concerned with real world industrial applications had considerable impact on my research. My sincere thanks go to Wolfram Büttner for his great

commitment to the concerns of the PhD students at Siemens, and to the whole CVE Team and all my colleagues at Siemens and Infineon. Many of them have provided invaluable help with all steps involved in putting together a thesis. This can only be a partial list, but I want to name Markus Kaltenbach, Thomas Rudlof, Rolf Drechsler and Raik Brinkmann, with whom I have closely worked together. Special thanks go to Michael Siegel for always sparing his time to discuss countless technical questions, for proof-reading and commenting of earlier versions, and for his advice on preparing the thesis and the conference papers.

Further thanks, which I also owe to Siemens Corporation and to Infineon Technologies, are for offering me the possibility of spending one and a half years of research work at their local subsidiaries in San Jose, California. Those 18 months have deeply influenced my life in many respects. Thanks go to the designers at Infineon CNP for introducing me to the field of hardware design and for sharing their insights in the design and simulation process. Special thanks to Gabriele Miller for emotional support and for showing me the best Sushi places in the Bay Area. Further thanks to Diana for providing donuts and bagels every Friday. Additionally, I thank Natarajan Shankar from SRI International for the invitation and the inspiring day at their office in Menlo Park.

Yet, this thesis would not be as it is without the help, support and friendship of lots of people who are not directly concerned with my work. Especially, and with all my heart, I want to express my gratitude to Peggy and Andy Gouw for embracing me and taking me in with all their warmth. I still consider you family! And Daniel, I keep hoping to meet you and see you compete at the 2004 Olympic Games!

I also thank Leighton Ridgard for being a friend from the first day we met and for being a magnificent badminton doubles partner. On all those trips to the gym and to the movies you reminded me that there are more things in life than just working. Further thanks to all the players at Sun Jose State and to the guys from Gunn High (Steve, John, Jonas, Russ, Steven, Percy, Ho, Jason, Jerome, ...) for the great games on challenge court on Sunday afternoons, keep on stroking! Thanks to all the members of the Sunnyvale Badminton Club (Ben, Andy, George, ...) and to all the kids I was coaching during that time (Eva, Jordan, Daniel, Michael, ...), I miss you all! And of course thanks to *Starbucks* for the *Extra Grande Caramel Latte Machiato*, to the *Mongolian Barbecue* and the *Kobe Restaurant* for excellent food, and to *Trader Joe's* for a life-saving supply of Milka chocolate.

Last, but certainly not least, I owe thanks to my family, my mom and dad, and to my friends for constantly giving support. A dearest thank you goes to Mirjana Ćosić for encouraging me and giving me the strength to go on, to Elke Siwochin, and to my old pal Michael 'Micky' Packhäuser, no matter how far apart, you were always there. In particular, I also want to express my very gratitude to Mirka Stroetzel who was the one who was there for me when I really wanted to give up and pack it all in.

Finally, I am obliged to Infineon Technologies for allowing me to take time off work in order to complete my degree.

Contents

Chapter 1

Introduction

Hardware verification has become one of the most important steps in digital circuit design. Today's hardware designs are highly integrated application specific circuits (ASICs), which are used in a wide-spread field of implementations, ranging from entertainment electronics and domestic appliances to business-relevant industrial production lines and safety-critical systems, like for example cars, airplanes or medical devices. Circuit designs need to be checked in order to ensure that manufactured chips operate correctly according to their specifications.

The average time-to-market for custom chip designs presently is about 18 months, and approximately between 60–70% of project costs result from error detection and correction. Production of faulty circuits requires redesigns and enlarges the number of design cycles. Each redesign of an averagely sized ASIC and each further design cycle causes additional costs of several hundred thousands of dollars. Efficient testing for correct behavior becomes more and more important and a major economical issue. It is desirable to detect design faults before manufacturing and thus to keep the number of design cycles and redesign costs as small as possible. Yet, fabricating hardware prototype chips, which can be used for testing already during the design phase, is time-consuming and expensive.

In the past, simulation was the common way to validate functionality of circuit designs, and still is today. But even for relatively small and plain designs exhaustive simulation almost always is impossible due to the immense number of input-stimuli that need to be checked. In practice, pure simulation can be used only for a partial validation of circuit functionality and cannot prove complete functional correctness.

Formal methods for functional hardware verification are becoming increasingly attractive in *Electronic Design Automation* (EDA), since they prove correct design behavior before manufacturing and without exhaustively simulating a design. Over the past years, a variety of formal verification techniques and methodologies has been proposed for hardware verification. However, only rather few techniques have shown to be suitable for application in an industrial environment. In the first place, this is caused by insufficient performance on large industrial designs, and secondly by the fact that the use of most techniques still requires expert knowledge of formal verification, which is normally not present in hardware design teams of chip manufacturing companies. Replacing simulation by formal verification especially is difficult if verification techniques

are not fully automated. Full automation of formal verification methods and the possibility for an easy integration into existing hardware design flows are required in order to be effectively applied in industrial hardware development processes, as well as a fairly ease of use and the capability of coping with real world design sizes.

Recently, *Bounded Model Checking* (BMC, see e.g. [BCC+99, BCCZ99, BCRZ01]), also referred to as *Property Checking* (see Chapter 2), has proven to be highly applicable for circuit verification. In BMC, a formal property is verified which describes an intended behavior of a given digital design within a finite bounded interval of time. BMC problems can be mapped onto instances of the satisfiability problem for Boolean formulae (SAT), which can be tackled by Boolean constraint solvers and Boolean function manipulation techniques. Recent advances in work on data structures, like *Binary Decision Diagrams* (BDDs, see e.g. [Bry86, Bry92, Bry95]), and advances in work on SAT solver technologies (see e.g. [LS01, MMM+01, Sht00, Sht01, SS96, SS99]) have made BMC a key technology to quality assurance in modern industrial hardware development processes, as is surveyed in [SS00]. The majority of today's most powerful industrial hardware verification tools is based upon multi-engine concepts of the above-mentioned technologies.

However, the SAT problem is known to be NP-complete, and design sizes are ever increasing. Nowadays, digital circuit designs frequently contain up to several million transistors. Verification of such large designs becomes more and more difficult, time-consuming and expensive. Formal verification methods are pushed to their limits, and verification tasks often cannot be completed due to design sizes and complexity issues, or simply cannot be executed because of resource limitations of today's computing machinery. Reducing runtimes and the amount of memory needed for computations is the priority objective in order to match today's sizes of real world designs in hardware verification. At the same time, ever improving verification techniques are required.

In this thesis a specific type of Bounded Model Checking problems is considered, and a new one-to-one abstraction technique for formal hardware verification of digital circuits is presented. The proposed abstraction technique exploits high-level design information in order to reduce the sizes of the BMC problems and thus to speed up the runtimes of SAT and BDD based hardware verification.

1.1 High-Level Information

Verification techniques for Bounded Model Checking of digital hardware are usually based on bit-level methods and bit-level formalisms (see [BCC+99, BCRZ01, SS00]). All variables occurring in a BMC problem are Boolean variables, representing single-bit signals of the circuit design. Circuit functionality is described by Boolean logic operations, and the verification task is solved in the Boolean domain. However, hardware designers normally specify digital circuits on a higher level of design abstraction, called *Word-Level* or *Register-Transfer-Level* (RTL). On this level of abstraction, collections of one-bit signals are grouped into semantic units (words, registers) with respect to circuit functionality. High-level data structures, like for example multi-bit signals, and high-level operators (i.e. operational units on high-level data structures) are available to describe circuit functionality.

Industrial design specifications are written in *Hardware Description Languages* (HDLs), like VHDL or Verilog, which allow to conceptualize the control flow and data flow of digital circuits on word-level. HDLs support common word-level data types (like for example registers), word-level data structures (like for example arrays and busses) and typical high-level operations performed in digital hardware, like for example shifting, addition and multiplication. A variety of standard tools is used in the design flow which read and further process HDL source code, like tools for simulation, timing analysis and physical layout. For verification purposes, HDL circuit design specifications must be transformed (*synthesis*) into bit-level models (e.g. Boolean formulae) before BMC techniques can be applied.

ASIC designs generally consist of hierarchies of building blocks. Often, designs, like for example ALUs, memory arrays or bus interfaces, have very regular structures that can easily be described on a higher level of abstraction. Likewise, higher levels of design abstraction allow an easy recognition of those structures. Complex structural information can conveniently be deduced from high-level specifications, whereas bit-level descriptions lack adequate means to syntactically reflect semantic information on such structural regularities. RTL specifications of digital circuits explicitly contain structural information which can be exploited to simplify hardware verification. This type of high-level information is not available in plain bit-level descriptions and cannot be utilized by verification tools if verification procedures operate on the basis of bit-level descriptions only. The information is lost during the transformation of RTL specifications into bit-level models while synthesis.

Recently, several approaches to formal circuit verification have been proposed which are based on RTL specification formalisms and which make use of structural high-level information. Examples are high-level verification techniques, like *Word-Level Decision Diagrams* (see e.g. [BC94, DBR97, CKRZ01, HD99, AH97, WAH01, Dre00]), *Integer Linear Programming* (e.g. [FDK98, BD02, ZKCR01, ZKC01]), *Symmetry Reductions* (e.g. [CEJS98, ET99]) and *Term Rewriting* (e.g. [DJ90]), to survey only a few. Other promising approaches to enhance the capabilities of existing formal verification tools are based on *Uninterpreted Functions* (see e.g. [BGV99, VB99, VB98, HIKB96, PRSS99]) or on *Automated Abstraction Techniques* (see e.g. [Joh01a, CGL92]).

1.2 Objectives of this Thesis

This thesis proposes a new word-level abstraction technique which speeds up property checking of digital hardware by reducing design sizes before verification. The proposed technique was first presented in [Joh01a]. Further details were presented in [Joh01b, Joh01c, JD01] and [JD02a, JD02b]. The technique exploits structural high-level information in RTL circuit specifications and automatically scales down the width of data path signals with respect to the property which is to be verified while trying to achieve a maximum degree of scaling.

1.2.1 Scaling Down Design Sizes by Signal Width Reduction

The core functionality of this approach is based on a simple and straightforward idea. High-level design specifications of digital circuits contain the structural information on how single bits are arranged to represent word-level signals and which individual bits belong to the same word-level

signal. The information about the widths of data path signals and about word-level data flow is available and can be exploited. Under specific conditions, it is possible to replace an n-bit data path of a circuit design by an m-bit data path, with $m < n$, and then to use the scaled and smaller version of the design for verification instead of the original one without altering verification results.

By now, data path scaling is a classical means for attacking the state space explosion problem. Reduction of data path widths is typically tried if verification of an ASIC which includes an n-bit data path takes too long or cannot be completed due to reasons of computational complexity, but up to now this technique has lacked full automation and a clear methodology on a formal basis. Still, many EDA companies today perform such reductions *manually* to speed up verification runtimes. Data path scaling is done based on experience and intuition of the circuit designer, very often without having the (formal) guarantee that the property which is considered and has to be verified really is independent of the width of the data path and that the chosen amount of scaling does not falsify verification results. Moreover, such manual modifications usually require intensive rewriting of the HDL code as shrinking the width of a data path causes additional side-effects. If, for example, a 32-bit bus is replaced by a bus of smaller width, say 16-bit, then the width of each signal which accesses the bus by read or write operations must be scaled too. Side-effects can go even farther if such a signal is the concatenation of several other smaller signals. Consider a design where a 24-bit signal reads information from the 24 most-significant bits of the bus while another 8-bit signal reads the 8 least-significant bits. At the outset, it is not clear how reducing the bus-width to 16 bits affects the two signals which read from the bus and how scaling has to be applied to them.

This thesis formalizes a new methodology which allows for a fully automated scaling of data path widths. The proposed technique efficiently analyzes word-level data flow in RTL descriptions with respect to a specified property. Designs are automatically scaled down by reducing signal widths before property checking, while guaranteeing that the property holds for the scaled model if and only if it holds for the original design. The reduced model of the circuit is used as input to existing hardware verification methods instead of the original design, thus speeding up property checking runtimes and allowing larger design sizes to be verified.

1.2.2 Minimum-Width Reductions of Bitvector Satisfiability Problems

The proposed abstraction technique is based on data flow specification by means of *formal bitvector theories* (see e.g. [CMR96, MR98]). Bitvector theories have proven to be an adequate means of describing digital hardware and related Bounded Model Checking problems at a higher level of design abstraction. Bitvectors are array-like structures of finite width over a two-valued domain, which can be used to model multi-bit circuit signals (see Chapter 3). Word-level data flow and control flow aspects of digital designs can be characterized by bitvector equations (see Chapter 5) in a way, such that design properties can be verified by determining satisfiability of such equations. Several decision procedures have been investigated which determine satisfiability or validity of bitvector equations, see e.g. [BDL96, BDL98, BP98, CMR96, CMR97, MR98, Joh99]. However, either the expressiveness of the term languages and the bitvector theories which are used is rather limited, or the performance of the decision procedures cannot compete with SAT and BDD based property checking when applied to large real world circuit designs.

This thesis proposes a different approach to deciding satisfiability of a system of bitvector equations. Instead of trying to directly solve the equations, the high-level information contained in the bitvector terms is used to compute a second corresponding and equivalent system of bitvector equations for which then satisfiability is determined by using conventional SAT and BDD based methods. Thus, a high-level abstraction technique for systems of bitvector equations is established, which is characterized in the way that the SAT problem which is related to the second system is smaller than the SAT problem related to the original system and therefore generally can be decided much faster and with less effort (see Figure 1.1 for an illustration of the basic concept).

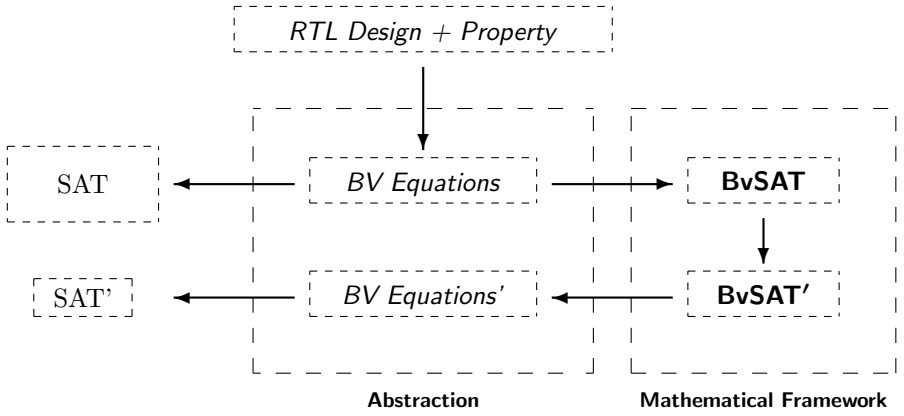


Figure 1.1: Basic Concept

One of the primary foci of this work is the investigation of satisfiability problems for an enhanced theory of fixed-size bitvectors which allows to describe complete real world BMC problems on RT-Level. We introduce the formal satisfiability problem **BvSAT** for bitvector functions and bitvector disequalities, which was first presented in [Joh01c] and which is a generalization of the SAT problem from Boolean variables to bitvectors of finite width (see Chapter 4). Satisfiability of systems of bitvector equations can be reduced to satisfiability of instances of **BvSAT** (see Chapter 6). The **BvSAT** problem constitutes the mathematical framework which we use to study satisfiability of systems of bitvector equations. We investigate in detail how satisfiability of **BvSAT** depends on the widths of the bitvector domains.

From the theoretical point of view, the main contribution of this thesis is a size reduction of **BvSAT** problems by means of a formal one-to-one correspondence between given instances of **BvSAT** and related instances over bitvector domains of *smaller* width. This correspondence maps satisfiable instances onto smaller satisfiable instances, and unsatisfiable ones onto smaller unsatisfiable instances. The correspondence is characterized by two theorems which prove correctness (i.e. the correspondence preserves satisfiability in a one-to-one fashion) and show optimality of the reductions (i.e. the correspondence yields minimal bitvector widths, see Chapter 4). The second contribution of this thesis is a high-level abstraction technique which utilizes the results stated by this correspondence in order to reduce a system of equations over bitvectors of certain widths into an equivalent system over bitvectors of smaller widths, while preserving satisfiability of the equations in a one-to-one fashion. The reduction is based on symbolic analysis

of word-level data flow and detection of uniform data dependencies. The proposed reduction technique furthermore provides an efficient way to compute satisfying solutions of the original system from satisfying solutions found for the reduced system.

Satisfiability of bitvector equations can be checked in the Boolean domain by transforming systems of bitvector equations into Boolean formulae, i.e. into instances of propositional SAT, and afterwards applying bit-level satisfiability checks, like SAT and BDD based procedures. Thus, bitvector formalisms are ideally suited for combining BMC and high-level verification techniques. The complexity of determining satisfiability of Boolean formulae generally depends on the number of Boolean variables occurring in the formulae. When systems of bitvector equations are transformed into SAT problems, Boolean variables are generated for each bit of each bitvector variable. Thus, the complexity of checking satisfiability of systems of bitvector equations directly depends on the overall number of bits of all input, internal and output signals occurring in a design, i.e. on the sum of the widths of all bitvectors occurring in the equations. As a consequence, width reductions can have a significant impact on the runtimes of satisfiability checkers as can be seen in the sections on experimental results in Chapter 10.

1.2.3 One-To-One Abstraction to Speed Up Verification

From the practical point of view, the main contribution of this thesis is a new and fully automated word-level abstraction technique, which operates as a preprocess for property checking of digital hardware (see Chapter 2). The proposed method computes a one-to-one RTL abstraction of a digital design in which the widths of word-level signals are reduced with respect to a property. Given an RTL specification of a digital design and a property φ, a reduced RTL model is generated in which each word-level signal $\boldsymbol{x}$ is replaced by a corresponding shrunken signal of width $m_{\boldsymbol{x}} \leq n$, where n denotes the original width of $\boldsymbol{x}$. The proposed technique establishes a one-to-one abstraction, i.e. φ holds for the original design if and only if φ holds for the reduced model. False-negatives cannot occur.

Design and abstract model differ from each other only as far as signal widths are concerned. Each $m_{\boldsymbol{x}}$ is the minimum number of bits for $\boldsymbol{x}$ which is necessary and sufficient in order to establish a one-to-one abstraction for φ and a reduced model of the abovementioned type. The width of each signal in the abstract RTL model is minimal with respect to the design, the property φ, and the abstraction technique we propose (i.e. by solely changing signal widths).

Furthermore, a post-processing of counterexamples is provided. If φ does not hold for the abstract RTL, and if a verification tool returns a counterexample in terms of value assignments to its input signals, then a counterexample for the original circuit is computed. The verification task itself is completely carried out on the scaled-down version of the design. If the property fails, then counterexamples for the original design are computed from counterexamples found on the reduced model (again see Chapter 2). The proposed method also strictly separates the pre-/postprocessing of design and counterexample and the property checking process itself. Thus, the approach is independent of the concrete realization of the property checker and can be combined with a variety of property checking techniques and can easily be integrated into existing verification flows. Moreover, if preprocessing yields that no reduction is possible for a given design and a property, then abstract model and original design are identical. Thus, the verification task itself is not impaired by using the proposed abstraction.

Linear signal width reductions result in exponentially smaller state spaces. A linear reduction of a signal's width from n bits down to m bits, $m < n$, causes an exponential reduction of the size of the induced state space from 2^n down to 2^m. Hence, the proposed abstraction to a great extent can have influence on the runtimes of verification tools and can significantly speed up verification. Furthermore, state space reductions allow larger design sizes to be verified. Experimental results on large industrial circuits have demonstrated the applicability and efficiency of the proposed method (see Chapter 10).

1.3 Annotated Table of Contents

The primary concern of this thesis is not hardware, but the proposal and formalization of a new verification methodology which automatically scales data path widths. Only very little knowledge about hardware is required as this work is not concerned with computer architecture on transistor level. Hardware is only considered at the behavioral level, and verification is always understood as functional verification. Throughout this thesis, the practical facets of the proposed method are used to motivate and illustrate the presentation of the theoretical aspects of this work. The remainder of this thesis comprises the following chapters:

Chapter 2 introduces the main aspects of Bounded Model Checking of digital circuits and the verification framework which is considered in this thesis. A short overview on high-level verification techniques is given, and the basic idea of signal width reductions is motivated. Furthermore, it is explained how the proposed abstraction technique is incorporated into existing standard BMC flows.

Chapter 3 gives a detailed introduction to the theory of fixed-size bitvectors and bitvector functions which form the mathematical background of the proposed abstraction technique. As a central item, the notion of *bitwise bitvector functions* is defined.

Chapter 4 introduces the formal satisfiability problem **BvSAT** for bitvector functions and bitvector disequalities. The central aspect, how satisfiability of instances of **BvSAT** problems is related to the width of the bitvector domains, is investigated in detail. A one-to-one correspondence with respect to satisfiability of bitwise **BvSAT** problems of different widths is pinpointed, and the fundamental theorems on width reductions are presented.

Chapter 5 defines bitvector terms and systems of bitvector equations, which provide a textual representation of bitvector functions and **BvSAT** problems. Systems of bitvector equations are used to represent the control flow and data flow of digital circuits and Bounded Model Checking problems on word-level.

Chapters 6, 7, 8 and **9** contain the computational details of the proposed abstraction technique. It is explained how satisfiability of a system of bitvector equations is reduced to satisfiability of a collection of **BvSAT** problems by syntactical analysis of the bitvector terms, and it is shown how the theoretical results on width reduction of Chapter 4 are applied to compute a modified system of bitvector equations which is equivalent (with respect to satisfiability) to the initial system and which represents a scaled RTL version of the original property checking problem. Chapter 8 in detail presents the most crucial aspect of the proposed abstraction technique, consisting of a separate handling of structural and dynamic data dependencies. This separation

greatly improves the reduction results and the amount of scaling that can be achieved. It is the reason why the proposed abstraction can effectually be applied to large real world circuit designs and Bounded Model Checking problems.

Chapter 10 contains case studies and experimental results which have been achieved on industrial chip designs and which document the successful practical application of the proposed method.

Chapter 11 concludes the thesis and considers some future directions of research.

Chapter 2

High-Level Property Checking of Digital Hardware

This chapter provides a detailed introduction to the hardware verification process and to the property checking framework which is considered in this thesis. It shall provide guidance and motivate the remaining chapters. The formal concepts of Bounded Model Checking and automated data path scaling are presented, and the idea of using high-level information to simplify verification is motivated.

2.1 Bounded Model Checking

The problem of verifying that digital circuit behavior is in accordance with the intention of the circuit designer is a problem of checking whether design behavior meets specific properties. The behavior of digital circuit designs is described by the variation of the values of input, output and internal signals over a period of time. Circuits which do not contain any data-storing components (such as registers, latches or memories) are called *combinational circuits*. Otherwise, they are called *sequential circuits*.

For functional simulation and verification, a discrete and linear model of time is used. Time is modeled as an infinite countable sequence of discrete points $t \in \mathbb{N}$ of time which represent the values of a global clock by which the timing of a circuit is determined. The period between two points of time t and $t+1$ is called a *clock cycle*. The input-output behavior of sequential digital circuit designs is formally described by *Mealy automata*. A Mealy automaton is a deterministic finite state-machine $M = (S, I, O, f, g)$ with a finite set S of *states*, a finite alphabet I of *input values*, a finite set O of *output values*, a *state-transition function* $f : I \times S \longrightarrow S$ and an *output function* $g : I \times S \longrightarrow O$.

In this thesis, a circuit design is characterized by a finite collection $\vec{i} = \langle i_1, \ldots, i_n \rangle$ of *input signals*, a finite collection $\vec{o} = \langle o_1, \ldots, o_m \rangle$ of *output signals*, and a finite collection $\vec{s} = \langle s_1, \ldots, s_p \rangle$ of *state signals* (or *registers*), where $i_1, \ldots, i_n$, $o_1, \ldots, o_m$ and $s_1, \ldots, s_p$ are bit-valued variables modeling the bits of the circuit signals. The sets of input values, output values, and of states

are then given by $I = \{0,1\}^n$, $O = \{0,1\}^m$ and $S = \{0,1\}^p$, and the functional behavior of the circuit is given by two vector-valued Boolean functions f and g which characterize state-transitions and output-values of the design for a single clock cycle. If $\vec{i}(t) = \langle i_1^t, \ldots, i_n^t \rangle$ are the values of the input signals at some point of time t, and $\vec{s}(t) = \langle s_1^t, \ldots, s_p^t \rangle$ are the states of the circuit registers at t, then the values $\vec{o}(t) = \langle o_1^t, \ldots, o_m^t \rangle$ of the output signals at t are given by

$$\vec{o}(t) \;=\; f(\,\vec{i}(t), \vec{s}(t)\,) \tag{2.1}$$

The next state values $\vec{s'}(t) = \langle s'^t_1, \ldots, s'^t_p \rangle$ of the registers at the end of the clock cycle are determined by

$$\vec{s'}(t) \;=\; g(\,\vec{i}(t), \vec{s}(t)\,) \tag{2.2}$$

The state space for all signals as a whole is finite; the collections $\vec{i}$, $\vec{o}$, $\vec{s}$ of Boolean variables and the two functions f, g are sufficient to completely specify the sequential functional behavior of the design. Figure 2.1 gives a schematic illustration of a functional circuit specification.

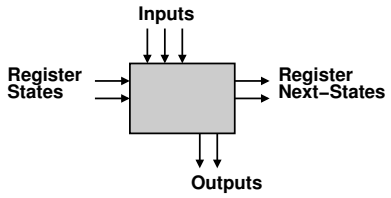


Figure 2.1: Functional Circuit Specification

A *bounded temporal property* defines an intended behavior of a design within a finite bounded interval of time $[t, t+c]$. A bounded temporal property can be specified by a propositional formula

$$\varphi(\;\vec{i}(t), \vec{o}(t), \vec{s}(t),\; \ldots\;, \vec{i}(t+c), \vec{o}(t+c), \vec{s}(t+c)\;) \tag{2.3}$$

over variables $\vec{i}(t), \vec{o}(t), \vec{s}(t), \ldots, \vec{i}(t+c), \vec{o}(t+c), \vec{s}(t+c)$ for the sequence of the circuit input, output and state signals for all clock cycles of the considered window of time, where $c \in \mathbb{N}_+$ is a constant and t is a free variable. Its semantics is given by a Boolean function

$$h_\varphi \;:\; (\,\{0,1\}^n \times \{0,1\}^m \times \{0,1\}^p\,)^{c+1} \;\longrightarrow\; \{0,1\} \tag{2.4}$$

If h_φ evaluates to true, then we say that the property holds for the regarded sequence of valuations of $\vec{i}$, $\vec{o}$ and $\vec{s}$ at $t, t+1, t+2, \ldots, t+c$, otherwise φ is said to fail. A schematic view of bounded temporal properties is given in Figure 2.2.

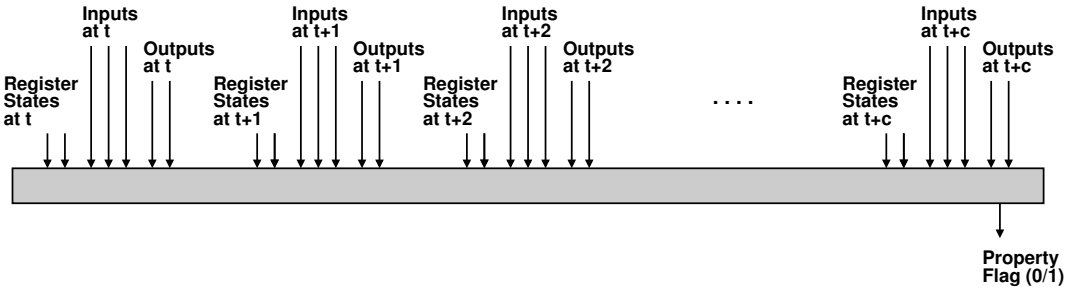


Figure 2.2: Bounded Temporal Property

Properties for Bounded Model Checking of digital circuits are usually specified in formal languages which describe propositional linear time logics over finite bounded intervals of time, similar to formulae used in *symbolic trajectory evaluation* (see e.g. [SHS97]). Such languages consist of bounded temporal operators and state expressions which can be augmented with time points and specify relationships among the values of registers, input and output signals. In practice, properties are often subdivided into an assumption part implying a commitment part:

$$\mathit{Property} \quad \equiv \quad \mathit{Assumptions} \implies \mathit{Commitments}$$

As a typical example consider:

```
assume:  (during [t, t+4]:  reset = 0) and
         (at t:  request = 1);
prove:   (at t+3:  acknowledge = 1) and
         (at t+4:  data = 11111111);
```

Hardware development is an error-prone process. It is likely that design specifications are not completely met by early implementations. Bounded Model Checking is intended to be applied already in early design stages in order to detect design errors as soon as possible. Instead of trying to *validate* a property φ, BMC performs a check for faulty behavior of a given design D (falsification of φ). Of interest is whether there exists a situation which contradicts φ, i.e. a sequence of values for all circuit signals of D for each clock-cycle of $[t, t+c]$ for which h_φ evaluates to false. The functional behavior of D within the window of time which is specified by φ depends on the sequence of values $\vec{i}(t), \vec{i}(t+1), \vec{i}(t+2), \ldots, \vec{i}(t+c)$ of the input signals, on the initial values $\vec{s}(t)$ of the states at the beginning of the time frame, and on the output function f and the state-transition function g of D. A sequence $\vec{i}(t), \vec{o}(t), \vec{s}(t), \ldots, \vec{i}(t+c), \vec{o}(t+c), \vec{s}(t+c)$ of values of circuit input, output and state signals describes a behavior of D if equations (2.1) and (2.2) and the next state relation

$$\vec{s}(t+1) = \vec{s'}(t) \tag{2.5}$$

hold for all clock cycles of $[t, t+c]$. This concept is called *unrolling* of the design and corresponds to creating separate instances $D_0, \ldots, D_c$ of the design D, one for each clock cycle, and then linking state and next state signals according to (2.5).

The initial property checking problem considers *sequential* design behavior. By unrolling the design it is turned into a problem for a *combinational* circuit. Compare Figures 2.1 and 2.3 for a visualization.

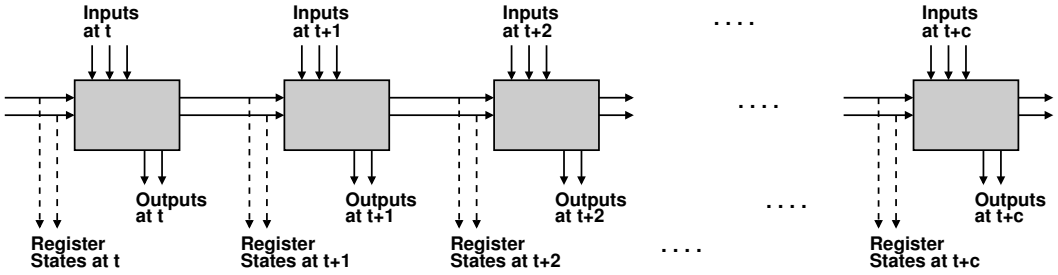


Figure 2.3: Unrolling

Whether or not the property φ holds for D can be determined by substituting the design specifications f and g into the negation of formula (2.3) according to equations (2.1), (2.2) and (2.5) for each clock cycle $t+i$ within the specified interval of time. The result is a propositional formula $\hat{\varphi}(\ \vec{i}(t), \vec{i}(t+1),\ \ldots\ , \vec{i}(t+c),\ \vec{s}(t)\ )$ in which only the state signals for the very first clock cycle and all input signals are free variables. The semantics of $\hat{\varphi}$ is then given by the Boolean function

$$h_{\hat{\varphi}}\ :\ (\ \{0,1\}^n\ )^{c+1} \times \{0,1\}^p\ \longrightarrow\ \{0,1\} \tag{2.6}$$

which is defined by

$$\begin{aligned} h_{\hat{\varphi}}(\ \vec{i}(t), \vec{i}(t+1),\ \ldots\ , \vec{i}(t+c),\ \vec{s}(t)\ ) = \\ \neg\ h_{\varphi}(\ \vec{i}(t), \vec{o}(t), \vec{s}(t),\ \ldots\ ,\ \vec{i}(t+c), \vec{o}(t+c), \vec{s}(t+c)\ ) \\ \wedge\ \ \vec{o}(t) = f(\ \vec{i}(t), \vec{s}(t)\ )\ \ \wedge\ \ \ldots\ \ \wedge\ \ \vec{o}(t+c) = f(\ \vec{i}(t+c), \vec{s}(t+c)\ ) \\ \wedge\ \ \vec{s'}(t) = g(\ \vec{i}(t), \vec{s}(t)\ )\ \ \wedge\ \ \ldots\ \ \wedge\ \ \vec{s'}(t+c) = g(\ \vec{i}(t+c), \vec{s}(t+c)\ ) \\ \wedge\ \ \vec{s}(t+1) = \vec{s'}(t)\ \ \wedge\ \ \ldots\ \ \wedge\ \ \vec{s}(t+c) = \vec{s'}(t+c-1) \end{aligned}$$

The formula $\hat{\varphi}$ is unsatisfiable if for all bounded windows of time of length c and for initial states $\vec{s}(t)$ and all input sequences $\vec{i}(t),\ \ldots\ , \vec{i}(t+c)$ the property φ holds for the design. If $\hat{\varphi}$ is satisfiable, then this is an indication that the circuit does not function in the way intended by the designer. Satisfiability of $\hat{\varphi}$ corresponds to a violation of the property:

$$\hat{\varphi} \text{ is satisfiable} \quad \Longleftrightarrow \quad \text{property } \varphi \text{ does not hold for design } D \tag{2.7}$$

A satisfying solution of $\hat{\varphi}$ yields value assignments for the state registers in the initial clock cycle t and a sequence of value assignments for all input signals in the clock cycles t to $t+c$. For this initial state $\vec{s}(t)$ and the input sequence $\vec{i}(t), \vec{i}(t+1), \vec{i}(t+2),\ \ldots\ , \vec{i}(t+c)$ a violation of the

property occurs. A satisfying solution of $\hat{\varphi}$ is called a *counterexample* for φ and D, and we say that property φ holds for D if $\hat{\varphi}$ is unsatisfiable. The question, whether or not $\hat{\varphi}$ is satisfiable, is called a *Bounded Model Checking Problem* (see [BCC+99, BCCZ99, BCRZ01]). The schematic relationship between design, property, unrolling and the Bounded Model Checking problem is shown in Figure 2.4.

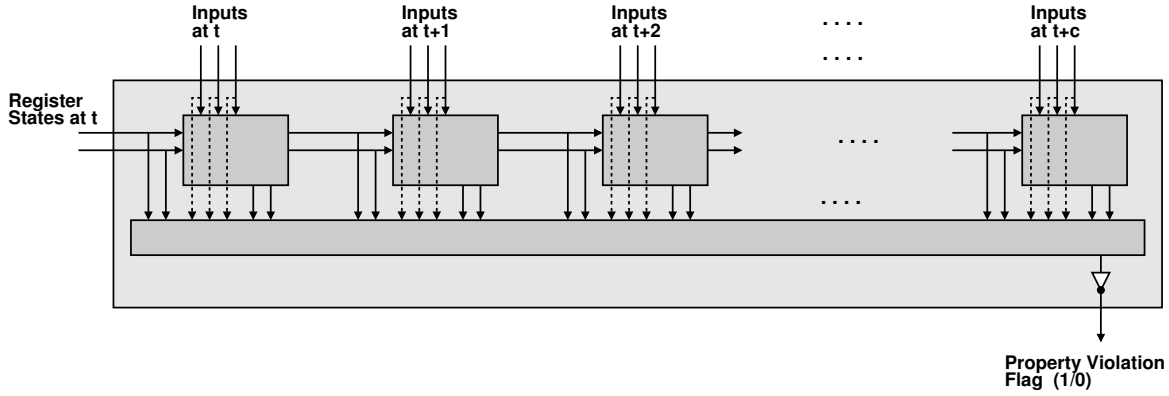


Figure 2.4: Bounded Model Checking Problem

Falsification is the preferred technique to find design errors as fast and as early as possible. Counterexamples yield assignments of values to the circuit inputs for which a violation of the desired behavior, which is described by the property, can be observed. Circuit designers use counterexamples for error-location and design debugging. If no counterexample exists, then the property holds for the design:

$$\text{design } D \text{ has property } \varphi \quad \Longleftrightarrow \quad \hat{\varphi} \text{ is unsatisfiable} \tag{2.8}$$

On the technical side, there are two main approaches to checking satisfiability of $\hat{\varphi}$ in Bounded Model Checking. In general, BDD techniques tend to prove unsatisfiability of $\hat{\varphi}$ more efficiently than SAT based techniques. Unsatisfiability corresponds to functional equivalence of $\hat{\varphi}$ with a constant false. In contrast to that, SAT solvers employ guided-search (see [Stå95, DP60]) and pruned search-space techniques and are better suited for quickly finding satisfying solutions of $\hat{\varphi}$ (see [BCCZ99, CFF+01, Sht00, Sht01]). Some prominent SAT solvers are, for example, Chaff (see [MMM+01]), SATO (see [Zha97]), Relsat (see [BS97]) and GRASP (see [Sil95, SS96, SS99]). Another successful technique for quickly finding satisfying solutions is *Automatic Test Pattern Generation* (ATPG), which is closely related to SAT techniques (see e.g. [HC00, Lar92, Che91, GAK99, CGPR90, BRTF99, SS97]). The majority of today's industrial hardware verification tools for Bounded Model Checking rely on multi-engine concepts of SAT, ATPG and BDD based techniques and try to partition the BMC problem in a divide-and-conquer approach into smaller problems. Based on heuristics, these smaller instances then are either split up again or tackled by BDD and SAT procedures. Computational limits depend on a tradeoff between circuit complexity, complexity of the logic of the properties, and the number of clock cycles specified in the property.

2.2 The Property Checking Framework

The Bounded Model Checking framework which is considered in this thesis is illustrated in Figure 2.5 and is characterized as follows.

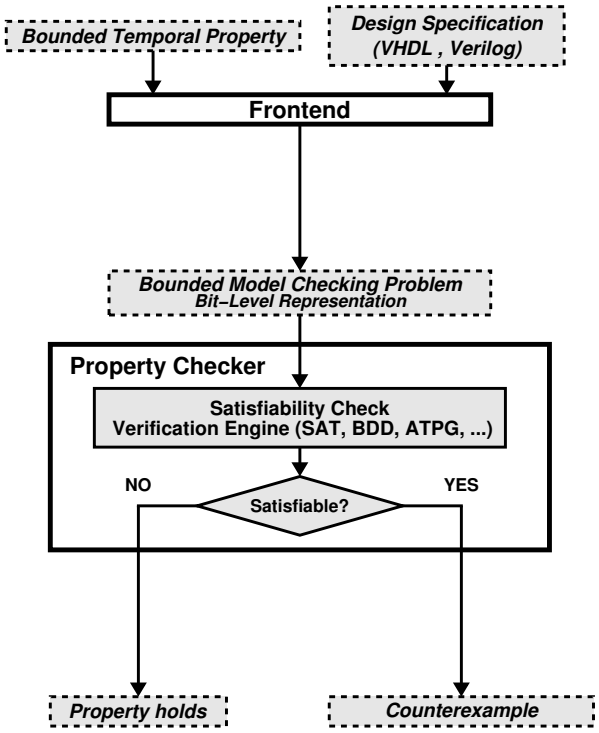


Figure 2.5: Property Checking Flow

Initially, the formal specification of a bounded temporal property and the HDL source code of a circuit design are read by a frontend which generates a bit-level representation of the corresponding Bounded Model Checking problem. The bit-level representation is then handed to a property checker, which determines satisfiability of the Bounded Model Checking problem and either, if the Bounded Model Checking problem is not satisfiable, confirms that the property holds, or, if the Bounded Model Checking problem is satisfiable, presents a counterexample in terms of value-assignments to the input signals of the design. The counterexample is generated from the satisfying solution that has been found and indicates a violation of the property.

2.3 Utilizing High-Level Information for Verification

Deciding satisfiability of Boolean formulae is known to be an NP-complete problem and has been well investigated. A variety of decision procedures has been proposed, and especially SAT procedures (see e.g. [Sil95]) have shown to be particularly useful in Bounded Model Checking (see also [SS00]). However, these procedures operate on bit-level. The problem, whether or

not a specific property holds for a circuit design, is solved in the Boolean domain. This means that all variables occurring in the Bounded Model Checking problem are Boolean variables. Each variable corresponds to a specific bit of one of the circuit signals. Yet, circuit designs are usually given in terms of RTL specifications, and RTL specifications contain explicit structural high-level information which is not contained in the Bounded Model Checking problem anymore and thus cannot be used in the satisfiability checks. The standard property checking flow thus includes a natural loss of structural information.

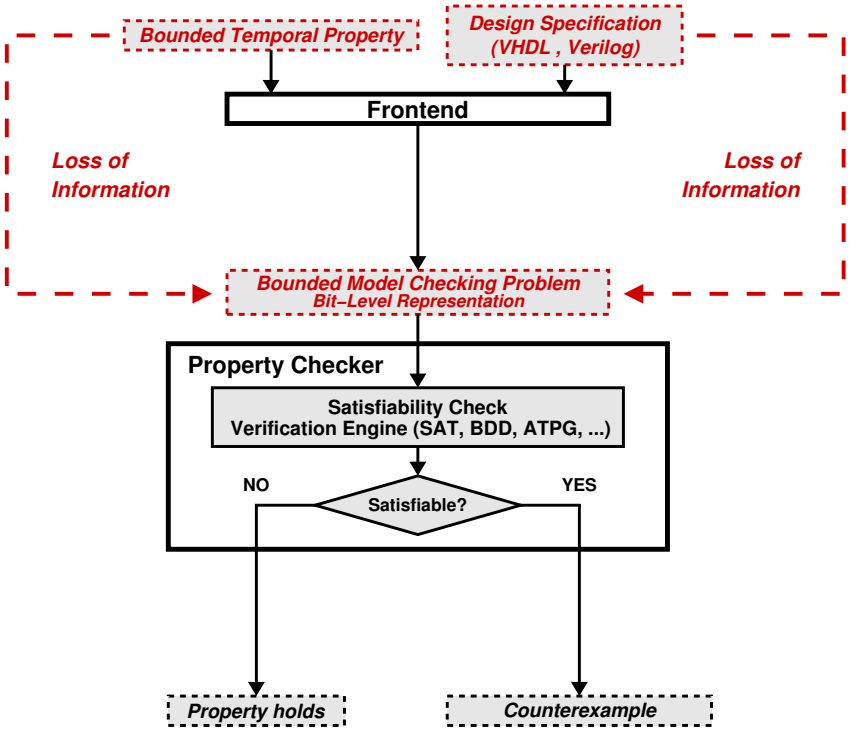


Figure 2.6: Word-Level Information vs. Bit-Level Information

Hardware description languages offer word-level data structures (e.g. multi-bit signals, arrays, memories) as well as high-level operators to describe circuit functionality on RT-Level. Many circuit designs have very regular structures that can easily be described on this level of abstraction. To give an example, consider addition of two 2-bit signals $\boldsymbol{x}_{[2]} = \langle x_1, x_0 \rangle$, $\boldsymbol{y}_{[2]} = \langle y_1, y_0 \rangle$, and a 1-bit carry-in $\boldsymbol{c}_{[1]} = \langle c_0 \rangle$. Sample bit-level equations describing the data path dependencies for the result signal $\boldsymbol{z}_{[2]} = \langle z_1, z_0 \rangle$ are:

$$\begin{aligned} z_0 &= (\,(x_0\overline{y_0} \vee \overline{x_0}y_0) \wedge \overline{c_0}\,) \vee (\,(x_0y_0 \vee \overline{x_0y_0}) \wedge c_0\,) \\ c_1 &= x_0y_0 \vee y_0c_0 \vee c_0x_0 \\ z_1 &= (\,(x_1\overline{y_1} \vee \overline{x_1}y_1) \wedge \overline{c_1}\,) \vee (\,(x_1y_1 \vee \overline{x_0y_1}) \wedge c_1\,) \\ c_2 &= x_1y_1 \vee y_1c_1 \vee c_1x_1 \end{aligned} \tag{2.9}$$

The connotation that, for example, x_1 and x_0 resemble a 2-bit signal, is not visible on bit-level and thus not visible to a bit-level satisfiability checker. Furthermore, it is rather difficult to

conclude from (2.9) that a 2-bit adder is represented. Contrasting that, 2-bit addition can easily be described on word-level by a single bitvector equation:

$$\boldsymbol{z}_{[2]} = \boldsymbol{x}_{[2]} + \boldsymbol{y}_{[2]} + \boldsymbol{c}_{[1]} \tag{2.10}$$

Such a representation allows special high-level decision procedures (here for high-level arithmetics, see e.g. [CZ95]) to be evoked by a verification tool. Several approaches to formal circuit verification are known which are based on word-level formalisms and which make use of RTL information. A selection of the most important ones is briefly surveyed in the following sections (see also [JD02b]). Several explanatory examples are given which illustrate that RTL and bit-level contain different levels of structural information and which motivate how additional word-level information can be used in order to simplify verification tasks for digital hardware.

2.3.1 Integer Linear Programming

One approach to RTL based verification is to treat arithmetic units by Integer Linear Programming (ILP) methods (see e.g. [ZKCR01, FDK98, BD02]). Word-level arithmetic is transformed into a linear program, i.e. a collection of arithmetic equations and inequalities over integer valued variables.

Linear programs are solved in the integer domain by ILP solvers. Equation (2.10), for example, can be transformed into the following ILP (note that solutions of Equation (2.11) and Equation (2.10) have a one-to-one correspondence):

$$\begin{aligned} &D = x + y + c \\ &D = 4 \cdot d + z \\ &0 \leq x < 4 \\ &0 \leq y < 4 \\ &0 \leq c < 2 \\ &0 \leq z < 4 \\ &0 \leq d \\ &0 \leq D \end{aligned} \tag{2.11}$$

Integer Linear Programming is suitable for data path verification of arithmetic circuits but lacks adequate modeling of the control path. Another drawback is that efficient ILP solvers often restrict integers and thus data path signals to be words of fixed, machinery-dependent width, e.g. 32 bits. Furthermore, bit-level addition and multiplication of n-bit signals in digital circuit corresponds to modulo-arithmetic in the ring $\mathbb{Z}_n$ and requires caution when generating the ILP (cf. [ZKC01]). In the example given above, the second equation and all the inequalities are used to model modulo-arithmetic on 2-bit bitvectors.

2.3.2 Rewriting

Term-rewriting techniques (see e.g. [BN98, DJ90]) can be used to simplify verification tasks. RTL rewriting can utilize high-level information on operators, which is not present on bit-level.

Let $\boldsymbol{x}_{[2]}, \boldsymbol{y}_{[2]}$ and $\boldsymbol{z}_{[2]}$ be signals of 2-bit width, and consider checking if the following equation

$$(\boldsymbol{x}_{[2]} + \boldsymbol{y}_{[2]}) + \boldsymbol{z}_{[2]} \quad = \quad \boldsymbol{y}_{[2]} + (\boldsymbol{z}_{[2]} + \boldsymbol{x}_{[2]}) \tag{2.12}$$

is satisfiable. Due to associativity and symmetry of addition, the left and right side of Equation (2.12) can be rewritten to

$$\boldsymbol{x}_{[2]} + \boldsymbol{y}_{[2]} + \boldsymbol{z}_{[2]} \quad = \quad \boldsymbol{x}_{[2]} + \boldsymbol{y}_{[2]} + \boldsymbol{z}_{[2]} \tag{2.13}$$

and easily be recognized as a tautology. However, a sample bit-level representation of (2.12) is

$$\begin{array}{lcl}
a_0 & = & (x_0\overline{y_0} \vee \overline{x_0}y_0) \\
c_{00} & = & x_0y_0 \\
a_1 & = & (\,(x_1\overline{y_1} \vee \overline{x_1}y_1) \wedge \overline{c_{00}}\,) \vee (\,(x_1y_1 \vee \overline{x_1y_1}) \wedge c_{00}\,) \\
c_{01} & = & x_1y_1 \vee y_1c_{00} \vee c_{00}x_0 \\
b_0 & = & (\,(z_0\overline{a_0} \vee \overline{z_0}a_0) \wedge \overline{c_{01}}\,) \vee (\,(z_0a_0 \vee \overline{z_0a_0}) \wedge c_{01}\,) \\
c_{10} & = & z_0a_0 \vee a_0c_{01} \vee c_{01}z_0 \\
b_1 & = & (\,(z_1\overline{a_1} \vee \overline{z_1}a_1) \wedge \overline{c_{10}}\,) \vee (\,(z_1a_1 \vee \overline{z_1a_1}) \wedge c_{10}\,) \\
d_0 & = & (z_0\overline{x_0} \vee \overline{z_0}x_0) \\
e_{00} & = & z_0x_0 \\
d_1 & = & (\,(z_1\overline{x_1} \vee \overline{z_1}x_1) \wedge \overline{e_{00}}\,) \vee (\,(z_1x_1 \vee \overline{z_1x_1}) \wedge e_{00}\,) \\
e_{01} & = & z_1x_1 \vee x_1e_{00} \vee e_{00}z_0 \\
f_0 & = & (\,(y_0\overline{d_0} \vee \overline{y_0}d_0) \wedge \overline{e_{01}}\,) \vee (\,(y_0d_0 \vee \overline{y_0d_0}) \wedge e_{01}\,) \\
e_{10} & = & y_0d_0 \vee d_0e_{01} \vee e_{01}y_0 \\
f_1 & = & (\,(y_1\overline{d_1} \vee \overline{y_1}d_1) \wedge \overline{e_{10}}\,) \vee (\,(y_1d_1 \vee \overline{y_1d_1}) \wedge e_{10}\,) \\
b_0 & = & f_0 \\
b_1 & = & f_1
\end{array} \tag{2.14}$$

introducing auxiliary variables for carry bits and for temporary results of the additions. Syntactically deducing equality of $\langle b_1, b_0\rangle$ and $\langle f_1, f_0\rangle$ from (2.14) is far from being easy. Term rewriting techniques are, for example, applied in proof systems which automatically deduce equality of terms, and in preprocessors which simplify high-level expressions before other high-level verification techniques are applied.

2.3.3 Symmetry Reductions

Another approach to exploit regular structures of high-level operators is to restrict the sizes of signal domains by symmetry reduction. Symmetries, for instance, occur when operator arguments are commutable. As an example, consider addition of 16-bit signals:

$$\boldsymbol{x}_{[16]} + \boldsymbol{y}_{[16]} \quad = \quad \boldsymbol{z}_{[16]} \tag{2.15}$$

Both signals $\boldsymbol{x}_{[16]}$ and $\boldsymbol{y}_{[16]}$ are used symmetrically with respect to addition. Here, without loss of generality, it is possible to assume $\boldsymbol{x}_{[16]} \leq \boldsymbol{y}_{[16]}$, because a satisfying solution of Equation (2.15) exists if and only if a satisfying solution exists which additionally satisfies $\boldsymbol{x}_{[16]} \leq \boldsymbol{y}_{[16]}$. Such

additional constraint on $\boldsymbol{x}_{[16]}$ and $\boldsymbol{y}_{[16]}$ can help to significantly decrease the size of search spaces in guided-search algorithms (cf. [Sil95]) as employed in most SAT procedures. Obviously, it is impossible to automatically deduce such constraints from bit-level representations like (2.14). For further details on symmetry reductions see e.g. [CEJS98, ET99, Bri01, CGLR96, ID93, CFJ93].

2.3.4 Word-Level Decision Diagrams

For representation and manipulation of Boolean functions, *Binary Decision Diagrams* (BDDs) have been proposed (see e.g. [Bry86, Bry92, Bry95]) and are used successfully in many industrial verification tools. Several extensions of BDDs to the word-level have been suggested. These approaches are based on *Word-Level Decision Diagrams* (WLDDs), which are graph-based representations of functions that allow for the representation of functions with a Boolean range and an integer domain (see e.g. [BC94, AH97, DBR97, HD98, HD99, Dre00, WAH01, CKRZ01]). WLDDs have received a lot of attention, since based on these data structures for the first time large arithmetic circuits, including multipliers, have been formally verified. But, in contrast to BDDs, the manipulation algorithms can be more expensive and often have exponential worst-case behavior.

2.3.5 Abstraction

Abstraction techniques (cf. e.g. [CGL92]) implement the following general approach. Instead of directly solving a given verification problem P, a smaller or simpler instance $P' := \mathcal{T}(P)$ is computed in which information that is not relevant for solving the verification problem is abstracted and which is then solved by conventional methods. Depending on the degree of reduction or simplification between P and P', solving P' can possibly be done faster and might require significantly less resources. It has to be ensured that computing P' from P preserves certain criteria as far as solvability is concerned. In this context, an abstraction technique $\mathcal{T}$ is said to be *one-to-one* if, for all problem instances P, solvability of $P' := \mathcal{T}(P)$ is related to solvability of P in a one-to-one fashion, i.e. if the abstract problem is solvable *if and only if* the original problem is solvable. Since we consider satisfiability problems, we have:

$$\mathcal{T} \text{ is a one-to-one abstraction:} \qquad (\, P' \text{ satisfiable} \iff P \text{ satisfiable} \,)$$

If the domains of abstract and original problem differ, then abstractions usually provide an additional transformation τ which computes solutions of the original problem from solutions found on the abstract problem instance, i.e. if s is a satisfying solution of $P' := \mathcal{T}(P)$, then $\tau(s)$ yields a satisfying solution of P. Thus, solving the original problem can completely and efficiently be replaced by solving the abstract problem instance, provided that the total amount of time for computing the abstract instance and then solving it is still faster than solving the original problem. For example, conversion of Boolean SAT formulae from CNF to DNF yields a one-to-one abstraction with respect to satisfiability. As far as complexity is concerned, deciding satisfiability of CNF formulae is NP-complete, whereas satisfiability of DNF formulae can be determined in polynomial time. However, whether there exists an efficient computation of the abstraction itself, is still an open problem which is equivalent to the P=NP problem.

If an abstraction is not one-to-one, then for each solution s found for P', an additional consistency check has to be performed which inspects if $\tau(s)$ indeed is a solution of P or not (a so-called *false-negative*). Such an abstraction still might be of interest if establishing a one-to-one abstraction is not possible, but finding solutions s of P' and performing the consistency check for $\tau(s)$ is fast. In such a case, abstraction is usually combined with guided-search techniques on the solution space of P'. For each solution that is found, a consistency check is performed, and the search is continued if this check fails. Yet, the amount of reduction of the problem size achieved by such an abstraction must justify the additional costs for validating solutions found for the abstract problem instance.

One-to-one abstractions are highly attractive in digital hardware verification because reduced or simplified problem instances can significantly increase the performance of existing verification tools. If the abstract problem instance is specified in the same formalism which is used for the original problem, then one-to-one abstractions can easily be embedded in existing verification flows without having to modify the underlying verification techniques. Additionally, abstractions which operate on RT-Level can incorporate and utilize all high-level information which is available in the problem specification.

2.4 Scaling Down RT-Level Designs

The improvement of formal verification of digital hardware which is contributed by this thesis consists of a high-level one-to-one abstraction technique which is used as a preprocessing step in the hardware verification flow. The proposed method exploits structural high-level design information and absorbs the informational gap between RTL and bit-level by establishing a fully automated scaling of the design.

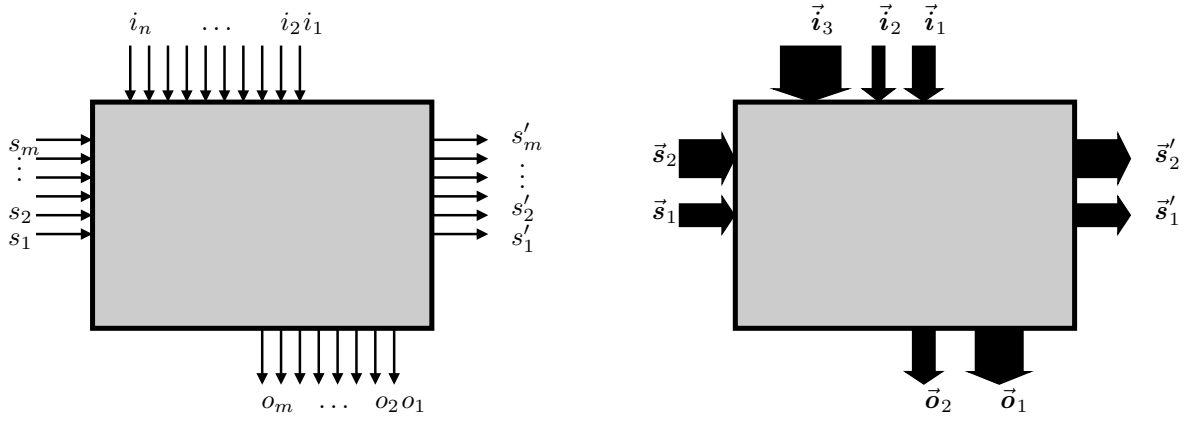


Figure 2.7: Bit-Level Design Specification vs. Word-Level Design Specification

Figure 2.7 demonstrates one of the differences between high-level and bit-level specifications of digital designs. The information which collections of single bits resemble high-level circuit signals is present in RTL specifications, but is lost on bit-level. As a consequence, the specification of

data flow on bit-level lacks this information, too. Data flow can only be specified in terms of single-bit signals and Boolean logic gates, whereas on RT-Level the data paths of a design are characterized by multi-bit busses and high-level operators and modules, as indicated in Figure 2.8.

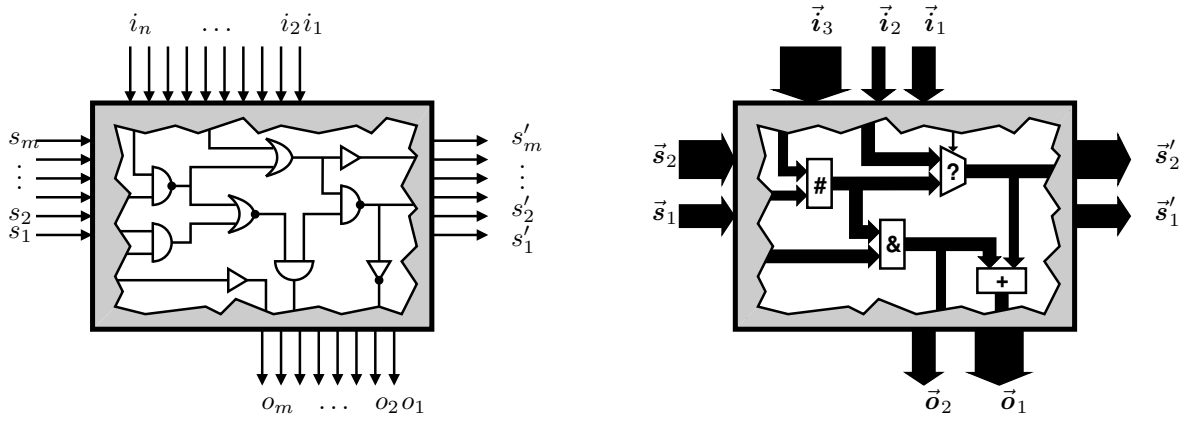


Figure 2.8: Bit-Level Data Flow vs. Word-Level Data Flow

The proposed abstraction technique exploits high-level information on multi-bit signals and high-level information on multi-bit data flow. In order to be able to do so, a prerequisite is to make this type of information available in an intermediate preprocessing stage of the verification flow. Therefore, in a first conceptual step, the conventional frontend is replaced by a new frontend which, instead of a bit-level representation, generates an RTL representation of the Bounded Model Checking problem in which the structural information on high-level data flow is preserved. Then, in a successive step, this RTL representation is further processed and transformed into the traditional bit-level representation, which is then handed to the property checker. The so modified verification flow is shown in Figure 2.9.

Up to this point, high-level information was only contained in the formal specification of the bounded temporal property and in the HDL design specification (cf. again Figure 2.6). Now, this information is combined and made available in a high-level representation of the Bounded Model Checking problem. Note that the loss of information is still inherent in the modified verification flow. It is deferred and now occurs in the newly introduced transformation step.

In this thesis, Bounded Model Checking problems are represented on RT-Level by formal systems of bitvector equations. For a given design D and a formal property φ the new frontend synthesizes a system E of bitvector equations such that the corresponding Bounded Model Checking problem $\hat{\varphi}$ is satisfiable if and only if E is satisfiable, i.e.

$$E \text{ is satisfiable} \quad \Longleftrightarrow \quad \text{property } \varphi \text{ does not hold for design } D \tag{2.16}$$

Bitvector terms, bitvector equations and systems of bitvector equations are introduced in detail in Chapter 5.

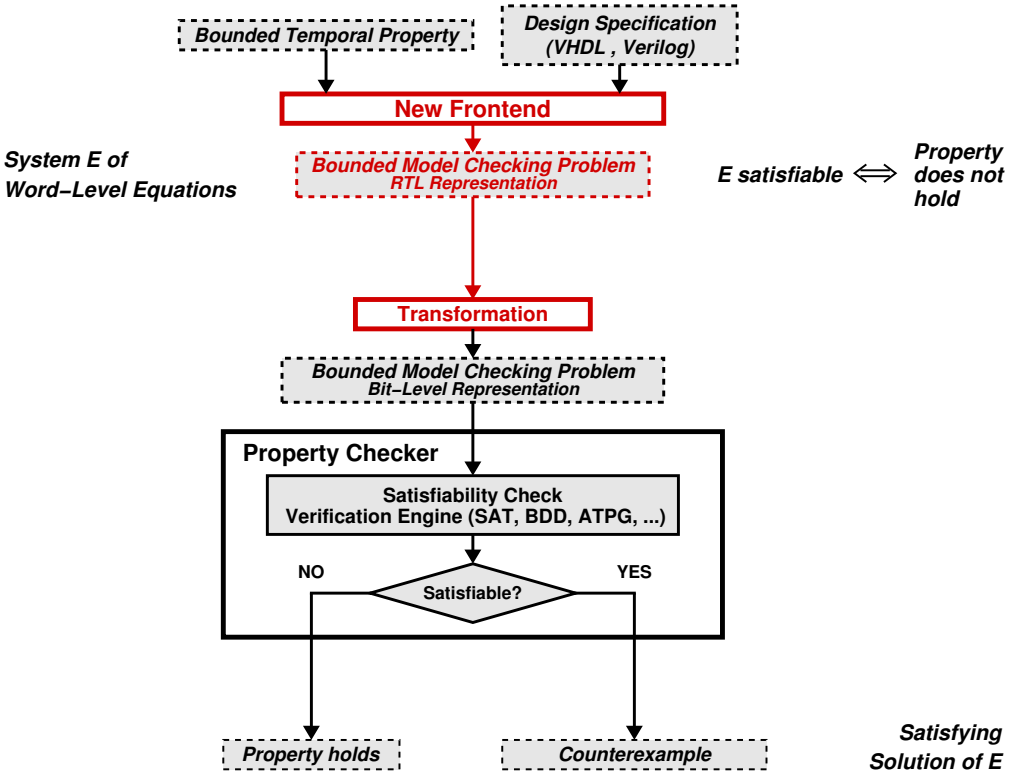


Figure 2.9: Modified Property Checking Flow

Bitvector variables of E correspond to multi-bit circuit signals of D. Each signal and each bitvector variable $\boldsymbol{x}$ has a fixed width $n \in \mathbb{N}_+$ (which usually is annotated as a subscript in square brackets) and takes bitvectors of respective length as values. The later transformation of E into a bit-level representation of the Bounded Model Checking problem generates one bit-level variable for each bit of a bitvector variable. Thus satisfying solutions of E directly correspond to satisfying solutions of $\hat{\varphi}$ and vice versa, and yield counterexamples for φ and D (see Figure 2.9). The proposed abstraction technique establishes a preprocessing step. The system E of bitvector equations is taken and analyzed, and a second system E' is computed which is then used for property checking instead of the original system E. The system E' is generated by replacing each word-level signal $\boldsymbol{x}$ of E by a corresponding shrunken signal of width $m \leq n$ (where n denotes the original width of $\boldsymbol{x}$).

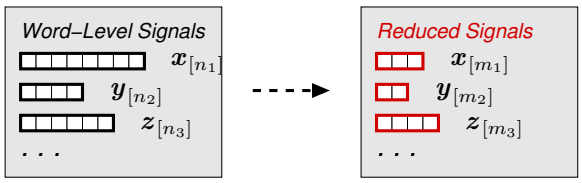


Figure 2.10: Basic Abstraction Technique

The original system E and the abstract model E' differ from each other only as far as signal widths are concerned. All other data flow aspects, like for example operators or term structure,

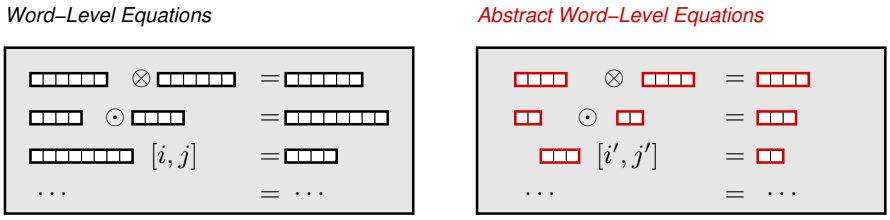


Figure 2.11: Reducing Systems of Word-Level Equations

are not changed (see Figure 2.11), and the proposed method computes the reduced widths such that satisfiability is strictly preserved, i.e.

$$E \text{ is satisfiable} \iff E' \text{ is satisfiable} \tag{2.17}$$

The width of each signal in the abstract model is the minimum width which is necessary and sufficient in order to establish a one-to-one abstraction with respect to design, property, and the proposed reduction (E' differing from E only by reduced variable widths). The reduced system E' then corresponds to a scaled version of the original design D, and scaling is understood in terms of strictly preserving the general data flow of D except for a reduction of the widths of the data paths, as illustrated in Figure 2.12.

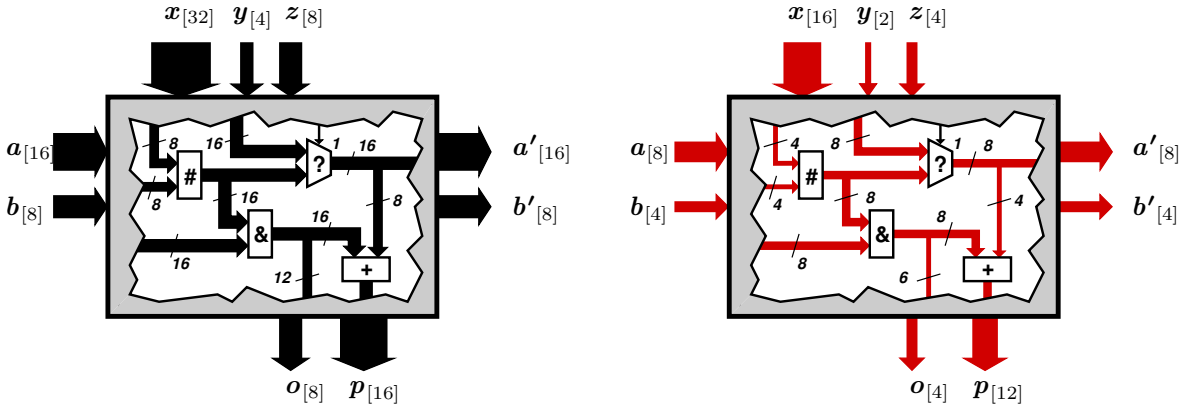


Figure 2.12: Original and Scaled Design

The amount of reduction that can be achieved is determined with respect to the circuit property that is to be verified. The scaled version of the design is then used for verification instead of the original one, and (2.17) yields:

the property holds for the original design $\iff$ the property holds for the scaled design

If the property does not hold, then, considering the modified verification flow, the counterexample which is returned by the property checker is a counterexample for the scaled version of

the design. The abstraction technique which is presented in this thesis adds an additional post-processing step to the verification flow. In this step, the reduced counterexample is taken and a counterexample for the original design is generated. The proposed reduction technique provides an easy generation of satisfying solutions of E from satisfying solutions of E'. Figure 2.13 illustrates how the proposed abstraction technique is integrated in the verification flow.

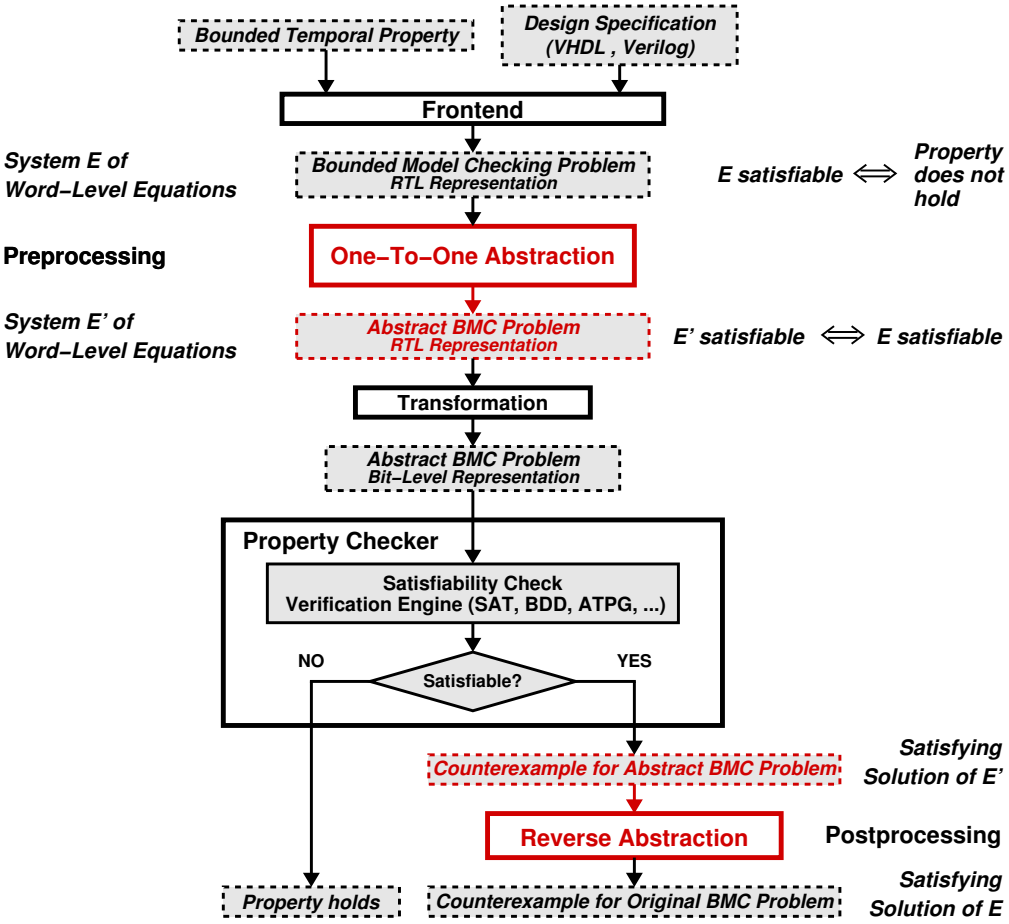


Figure 2.13: Property Checking Flow with High-Level Abstraction

The process of scaling down signal widths is separated into two subsequent phases. First, the coarsest *granularity* of each word-level signal is computed as determined by the structural data dependencies of E. A granularity is a separation of a signal into several contiguous parts which indicate the coarsest possible subsumptions of individual bits of the signal which are processed on the same data path. Details are given in Chapter 6 (see also Example 2.3). Then, for each such part, the necessary minimum width is computed which guarantees that satisfiability of E and E' correspond to each other in a one-to-one fashion. This will be further explained in Chapter 4. According to these computed minimum widths, the reduced width for the corresponding signal is reassembled. The basic concept of this technique is shown in Figure 2.14 (see also Chapter 9).

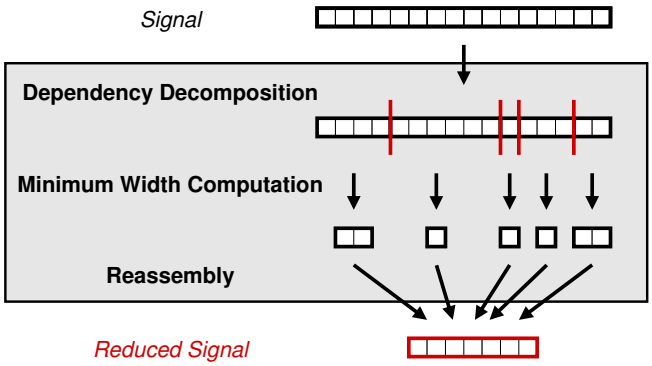


Figure 2.14: Signal Width Reduction

The bit-level representation of the Bounded Model Checking problem which is generated from the RTL representation contains bit-level variables for each bit of each word-level signal. Depending on the degree of reduction of the signal widths during scaling, the bit-level representation can contain significantly less variables when the abstract RTL model is used (see Figure 2.15).

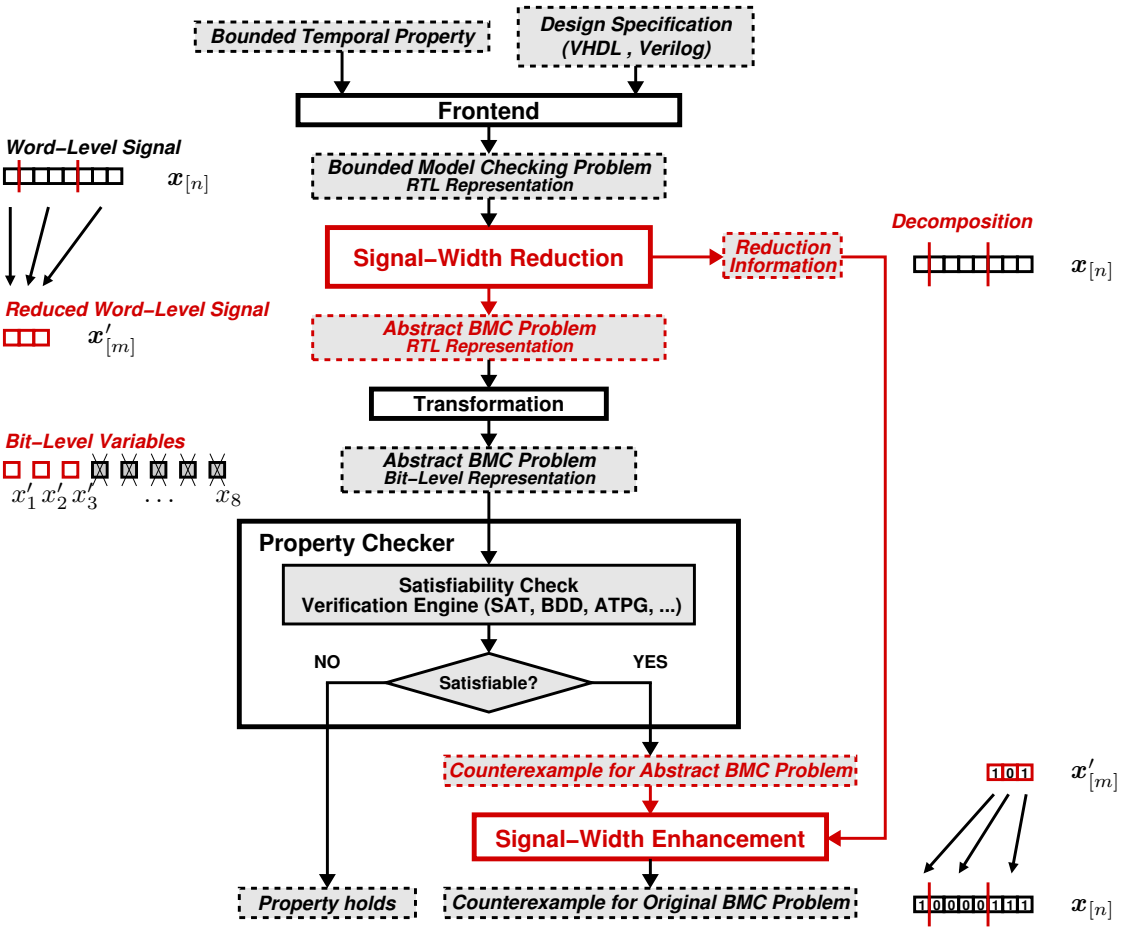


Figure 2.15: Speeding Up Property Checking by Automated Data Path Scaling

The effect is a reduction of the sizes of the Bounded Model Checking problems which have to be handled by the property checker. This aspect coincides with a speed-up of the verification runtimes. Thus, although property checking is still done on bit-level, this verification approach indirectly uses and benefits from high-level information. Another advantage is that no modifications have to be applied to the property checker; the proposed abstraction can easily be integrated into existing flows (see also [JD01, JD02a] for an overview).

2.5 Signal Width Reduction

Systems of bitvector equations can be used to describe data flow and control flow aspects of a digital circuit on word-level (bitvector terms and bitvector equations are formally defined in Chapter 5). Design properties can be verified by checking satisfiability of such equations. The high-level information about which individual bits belong to the same multi-bit high-level signal and the high-level information about how single bits are grouped and ordered within these signals is explicitly contained in the equations. In the following introductory examples the basic idea for using this information to reduce computational complexity of satisfiability checks of bitvector equations is illustrated.

Example 2.1 (Uniform Data Dependencies) *Determining if bitwise Boolean conjunction of two word-level signals of width 8, denoted by* $\boldsymbol{x}_{[8]}$ *and* $\boldsymbol{y}_{[8]}$*, can evaluate to the 8-bit zero vector can be done by checking if the following bitvector equation*

$$\boldsymbol{x}_{[8]} \text{ and } \boldsymbol{y}_{[8]} \;=\; 00000000 \tag{2.18}$$

is satisfiable, where and *denotes bitwise Boolean conjunction. Equation (2.18) specifies functional data dependencies between* $\boldsymbol{x}_{[8]}$ *and* $\boldsymbol{y}_{[8]}$*, and satisfiability of (2.18) is characterized by satisfiability of the following corresponding bit-level formula*

$$(x_0 \text{ and } y_0 \;=\; 0) \;\wedge\; (x_1 \text{ and } y_1 \;=\; 0) \;\wedge\; \ldots \;\wedge\; (x_7 \text{ and } y_7 \;=\; 0) \tag{2.19}$$

involving 16 Boolean variables and 8 bit-level equations. Obviously, it is not necessary to solve all 8 equations of (2.19) separately because bit-positions 0–7 of $\boldsymbol{x}_{[8]}$ *and* $\boldsymbol{y}_{[8]}$ *are treated uniformly, i.e. the data dependencies for* x_0, y_0 *to* x_7, y_7 *are the same except for index numbering. Let* $\boldsymbol{x'}_{[1]}$ *and* $\boldsymbol{y'}_{[1]}$ *denote two new word-level signals of width 1, derived from the variables of (2.18). It is sufficient to check if*

$$\boldsymbol{x'}_{[1]} \text{ and } \boldsymbol{y'}_{[1]} \;=\; 0 \tag{2.20}$$

is satisfiable as (2.18) is satisfiable if and only if (2.20) is satisfiable. Furthermore, a satisfying solution of (2.18) can be obtained from a satisfying solution of (2.20) by copying the values of $\boldsymbol{x'}_{[1]}$ *and* $\boldsymbol{y'}_{[1]}$ *into all bit positions of the corresponding signals of (2.18). For example,* $\boldsymbol{x'}_{[1]} = 0$ *and* $\boldsymbol{y'}_{[1]} = 1$ *yields* $\boldsymbol{x}_{[8]} = 00000000$ *and* $\boldsymbol{y}_{[8]} = 11111111$*.* □

In Example 2.1, both signals $\boldsymbol{x}_{[8]}$ and $\boldsymbol{y}_{[8]}$ have uniform data dependencies and can be reduced to a width of one bit. Uniform data dependencies are formally introduced in Section 3.4 and form the basis of the width reduction technique, as explained in Chapter 4. In general, the possible amount of reduction depends on structural and dynamical data dependencies as well, and the data flow of a signal must be analyzed for all equations of a given system of bitvector equations. Reduction depends on the data dependencies imposed by the conjunction of all equations. Thus, even if uniform data flow exists for specific signals in one equation, other equations can be the reason that reduction to only 1-bit width might not preserve satisfiability in a one-to-one fashion.

Example 2.2 (Dynamical Data Dependencies) *Let $\boldsymbol{x}_{[8]}$, $\boldsymbol{y}_{[8]}$ and $\boldsymbol{z}_{[8]}$ be signals of 8-bit width for which uniform data dependencies exist. Consider a system of bitvector equations which additionally contains equations involving the following expressions:*

$$\begin{aligned} \ldots &= \mathsf{if}\ (\boldsymbol{x}_{[8]} = \boldsymbol{y}_{[8]})\ \mathsf{then}\ \ldots\ \mathsf{else}\ \ldots \\ \ldots &= \mathsf{if}\ (\boldsymbol{y}_{[8]} = \boldsymbol{z}_{[8]})\ \mathsf{then}\ \ldots\ \mathsf{else}\ \ldots \\ \ldots &= \mathsf{if}\ (\boldsymbol{z}_{[8]} = \boldsymbol{x}_{[8]})\ \mathsf{then}\ \ldots\ \mathsf{else}\ \ldots \end{aligned} \tag{2.21}$$

A satisfying solution of (2.21) might only exist if the values of $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]}$ are mutually different, i.e.

$$\boldsymbol{x}_{[8]} \neq \boldsymbol{y}_{[8]} \ \wedge\ \boldsymbol{y}_{[8]} \neq \boldsymbol{z}_{[8]} \ \wedge\ \boldsymbol{z}_{[8]} \neq \boldsymbol{x}_{[8]}\,. \tag{2.22}$$

Then, reduction to only one bit width is not possible, because

$$\boldsymbol{x'}_{[1]} \neq \boldsymbol{y'}_{[1]} \wedge \boldsymbol{y'}_{[1]} \neq \boldsymbol{z'}_{[1]} \wedge \boldsymbol{z'}_{[1]} \neq \boldsymbol{x'}_{[1]} \tag{2.23}$$

is not satisfiable, while (2.22) is. Instead the following holds:

$$\boldsymbol{x}_{[m]} \neq \boldsymbol{y}_{[m]} \ \wedge\ \boldsymbol{y}_{[m]} \neq \boldsymbol{z}_{[m]} \ \wedge\ \boldsymbol{z}_{[m]} \neq \boldsymbol{x}_{[m]} \tag{2.24}$$

is satisfiable for all $m \geq 2$, and at the same time 2 is the minimum value for m for which

$$\text{(2.22) satisfiable} \iff \text{(2.24) satisfiable}$$

holds. Therefore, satisfiability of (2.21) can be preserved by choosing reduced bitvectors of width 2. But, even if instead of (2.22) a solution of (2.21) just requires the weaker condition

$$\boldsymbol{x}_{[8]} \neq \boldsymbol{y}_{[8]} \ \wedge\ \boldsymbol{y}_{[8]} \neq \boldsymbol{z}_{[8]} \tag{2.25}$$

a reduction to 1-bit width still might not work, although $\boldsymbol{x'}_{[1]} \neq \boldsymbol{y'}_{[1]} \wedge \boldsymbol{y'}_{[1]} \neq \boldsymbol{z'}_{[1]}$ is satisfiable. The reason is that further equations of (2.21) can impose additional uniform data dependencies between $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]}$. For example, consider the additional equation

$$\mathtt{11111111} \ =\ (\ \boldsymbol{x}_{[8]}\ \mathsf{and}\ \boldsymbol{y}_{[8]}\ \mathsf{and}\ \mathsf{neg}(\boldsymbol{z}_{[8]})\ )\ \mathsf{or}\ (\ \mathsf{neg}(\boldsymbol{x}_{[8]})\ \mathsf{and}\ \boldsymbol{y}_{[8]}\ \mathsf{and}\ \boldsymbol{z}_{[8]}\ ) \tag{2.26}$$

The conjunction of (2.25) and (2.26) is satisfiable for bitvectors of width 8, but the corresponding problem in which variables are reduced to 1-bit width is unsatisfiable. Instead, satisfiability again is preserved when reduced bitvectors of width 2 are chosen. □

Dynamical data dependencies are thoroughly investigated in Chapter 8. Another important aspect of bitvector equations is that data dependencies can exist between complete word-level signals or only between certain bits. Typically, different data dependencies exist for different chunks of a variable. Variables can always be partitioned into contiguous parts in which all bits are treated uniformly with respect to data dependencies (this is further explained in Chapters 6).

Example 2.3 (Structural Data Dependencies) *Let $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}$ and $\boldsymbol{z}_{[8]}$ be bitvector variables of width 8, and let $\boldsymbol{a}_{[2]}, \boldsymbol{b}_{[6]}$ be bitvector variables of width 2 and 6. Let $\otimes$ denote concatenation of bitvectors, and consider the following system E of bitvector equations:*

$$E \left\{ \begin{array}{lllll} \boldsymbol{x}_{[8]} & \mathsf{and} & \boldsymbol{y}_{[8]} & = & \boldsymbol{z}_{[8]} \\ \boldsymbol{x}_{[8]} & & & = & \boldsymbol{a}_{[2]} \otimes \boldsymbol{b}_{[6]} \end{array} \right. \tag{2.27}$$

The first equation specifies uniform data dependencies for $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]}$, but the second one imposes different structural dependencies for the upper two and the lower six bits of $\boldsymbol{x}_{[8]}$. E can be decomposed into two disjoint independent systems E_1 and E_2 of bitvector equations,

$$\begin{array}{lll} E_1 & \left\{ \begin{array}{lllll} \boldsymbol{x}_{[8]}[7,6] & \mathsf{and} & \boldsymbol{y}_{[8]}[7,6] & = & \boldsymbol{z}_{[8]}[7,6] \\ \boldsymbol{x}_{[8]}[7,6] & & & = & \boldsymbol{a}_{[2]} \end{array} \right. \\ E_2 & \left\{ \begin{array}{lllll} \boldsymbol{x}_{[8]}[5,0] & \mathsf{and} & \boldsymbol{y}_{[8]}[5,0] & = & \boldsymbol{z}_{[8]}[5,0] \\ \boldsymbol{x}_{[8]}[5,0] & & & = & \boldsymbol{b}_{[6]} \end{array} \right. \end{array} \tag{2.28}$$

such that the set of satisfying solutions of E is the same as the set of satisfying solutions of the conjunction of E_1 and E_2, i.e. E is satisfiable if and only if $E_1 \wedge E_2$ is satisfiable. Furthermore, all data dependencies in E_1 and in E_2 are uniform. Satisfiability of (2.28) then can be reduced to satisfiability of:

$$\begin{array}{lll} E_1' & \left\{ \begin{array}{lllll} \boldsymbol{x'}_{[1]} & \mathsf{and} & \boldsymbol{y'}_{[1]} & = & \boldsymbol{z'}_{[1]} \\ \boldsymbol{x'}_{[1]} & & & = & \boldsymbol{a'}_{[1]} \end{array} \right. \\ E_2' & \left\{ \begin{array}{lllll} \boldsymbol{x''}_{[1]} & \mathsf{and} & \boldsymbol{y''}_{[1]} & = & \boldsymbol{z''}_{[1]} \\ \boldsymbol{x''}_{[1]} & & & = & \boldsymbol{b''}_{[1]} \end{array} \right. \end{array} \tag{2.29}$$

and from (2.29) we can recompose

$$E' \left\{ \begin{array}{lllll} \boldsymbol{x'''}_{[2]} & \mathsf{and} & \boldsymbol{y'''}_{[2]} & = & \boldsymbol{z'''}_{[2]} \\ \boldsymbol{x'''}_{[2]} & & & = & \boldsymbol{a'''}_{[1]} \otimes \boldsymbol{b'''}_{[1]} \end{array} \right. \tag{2.30}$$

where (2.30) is satisfiable if and only if (2.27) is satisfiable, and where $\boldsymbol{a'''}_{[1]}$ relates to $\boldsymbol{a}_{[2]}$, $\boldsymbol{b'''}_{[1]}$ relates to $\boldsymbol{b}_{[6]}$, $\boldsymbol{x'''}_{[2]}[1,1]$ relates to $\boldsymbol{x}_{[8]}[7,6]$, and $\boldsymbol{x'''}_{[2]}[0,0]$ to $\boldsymbol{x}_{[8]}[5,0]$, and so on. To obtain a solution of (2.27), signed extension is done separately for related chunks according to the prior decomposition, for example $\boldsymbol{a'''}_{[1]} = \mathtt{0}$, $\boldsymbol{b'''}_{[1]} = \mathtt{1}$, $\boldsymbol{x'''}_{[2]} = \mathtt{01}$, $\boldsymbol{y'''}_{[2]} = \mathtt{11}$, $\boldsymbol{z'''}_{[2]} = \mathtt{01}$, yields $\boldsymbol{a}_{[2]} = \mathtt{00}$, $\boldsymbol{b}_{[6]} = \mathtt{111111}$, $\boldsymbol{x}_{[8]} = \mathtt{00111111}$, $\boldsymbol{y}_{[8]} = \mathtt{11111111}$, $\boldsymbol{z}_{[8]} = \mathtt{00111111}$. □

In the following chapters, we will now formally define bitvectors and standard high-level operations on bitvectors (Chapter 3). The formal satisfiability problem **BvSAT** for bitvectors of fixed-width is introduced (Chapter 4), constituting the mathematical framework which is used to characterize satisfiability of systems of bitvector equations (Chapter 5). We prove that satisfiability of pure uniform data dependencies can always be described using only single-bit bitvector variables, as seen in Example 2.1. This is done by a general investigation of how satisfiability of specific **BvSAT** problems can be reduced to satisfiability of **BvSAT** problems over bitvector domains of smaller width (Chapter 4). The results of this investigation are used to establish a one-to-one abstraction technique for systems of bitvector equations. We show (Chapters 6 and 7) how a reduced width for each bitvector variable occurring in a system of equations is computed (cf. Figure 2.14), taking into account all structural (Chapter 6) and dynamical (Chapter 8) data dependencies, as illustrated in Examples 2.3 and 2.2. We furthermore show how a corresponding system of bitvector equations is generated with respect to these computed bitvector widths (Chapter 9).

Chapter 3

Fixed-Size Bitvectors and Bitvector Functions

In this chapter, the formal framework of the proposed abstraction technique is introduced. The following sections comprise fundamental definitions and preliminaries. The basic terminology which is used throughout this thesis is presented. As a first step, the notion of *fixed-size bitvectors* is addressed. Fixed-size bitvectors are the primal data type which is used for modeling circuit signals of digital hardware on *Register-Transfer-Level* (RTL). Section 3.1 explains the mathematical background and shows how values of circuit signals are represented by bitvectors. As a second step, section 3.2 introduces *bitvector functions*. Bitvector functions are mathematical objects which are used to formally describe the input-output behavior of digital circuits. Bitvector functions can be employed as functional specifications of RTL operators and RTL data flow between circuit units. In Section 3.3, a standard selection of important RTL operators is presented. Sections 3.4 and 3.5 deal with a special sub-class of bitvector functions, called *bitwise bitvector functions*, which provide an adequate way to characterize uniform data flow aspects of RTL operators upon which the provided abstraction technique is based.

3.1 Fixed-Size Bitvectors

Fixed-Size bitvectors are a special case of common array-like data structures which hold elements of a two-valued domain. The vector elements are called *bits*, and the possible values for bits are represented by zero and one.

Definition 3.1 (Bitvalues) *Let* $\mathbb{B} := \{\mathtt{0}, \mathtt{1}\}$ *denote the set of* ***bitvalues*** $\mathtt{0}$ *and* $\mathtt{1}$. ■

For each $v \in \mathbb{B}$, let $\overline{v}$ denote the opposite (*inverted*) bitvalue, i.e. $\overline{\mathtt{0}} := \mathtt{1}$ and $\overline{\mathtt{1}} := \mathtt{0}$. Bitvectors are finite vectors of bits. The width of a bitvector is constrained to be a positive natural number, bitvectors of width 0 are excluded. Furthermore, a linear ordering on the positions of the vector elements is assumed. Let $\mathbb{N}_+$ denote the set of positive natural numbers, i.e. the set $\mathbb{N}_{>0}$ of natural numbers greater than zero.

Definition 3.2 (Fixed-Size Bitvectors) *Let $n \in \mathbb{N}_+$. A **bitvector of width** n is a vector element $\langle v_{n-1}, \ldots, v_1, v_0 \rangle \in \mathbb{B}^n$, consisting of n individual bits which are indexed from right to left, starting with index position 0.* ■

If $\langle v_{n-1}, \ldots, v_0 \rangle$ is a bitvector of width $n \in \mathbb{N}_+$, then v_{n-1} is called the *most significant bit* and v_0 is called the *least significant bit*.

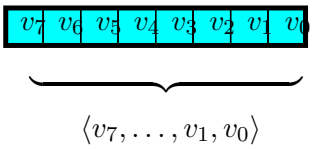


Figure 3.1: Illustration of Fixed-Size Bitvectors

Fixed-size bitvectors are the primal data type which is used throughout this thesis. For each $n \in \mathbb{N}_+$, let the set of fixed-size bitvectors of width n be denoted by:

$$\mathbb{B}_{[n]} \quad := \quad \underbrace{\mathbb{B} \times \ldots \times \mathbb{B}}_{n}$$

The set of all bitvectors of finite width is denoted by:

$$\mathbb{B}_{[*]} \quad := \quad \bigcup_{n \in \mathbb{N}_+} \mathbb{B}_{[n]}$$

We define a special kind of typed variables, called *bitvector variables*, which will be used to represent fixed-size bitvectors. Each bitvector variable can only take bitvectors of a fixed specific width as values. The type of a bitvector variable indicates the width of the bitvectors which can be taken as values. The type is determined by a fixed (but arbitrary) well-known positive natural number $n \in \mathbb{N}_+$, which is annotated to the variable as a lower index in square brackets.

Definition 3.3 (Bitvector Variables) *Let $n \in \mathbb{N}_+$. A **bitvector variable of width** n is a typed variable $\boldsymbol{x}_{[n]}$ representing fixed-size bitvectors of width n.* ■

Bitvector variables will be denoted by bold face characters. We will write $\boldsymbol{x}_{[n]}[i]$ to refer to the value v_i of the i^{th} bit of a bitvector $\boldsymbol{x}_{[n]} \in \mathbb{B}_{[n]}$ with $\boldsymbol{x}_{[n]} = \langle v_{n-1}, \ldots, v_1, v_0 \rangle$.

Note: We explicitly allow usage of the same symbol (name) for bitvector variables of different widths. If $n, m \in \mathbb{N}_+$ with $n \neq m$, then $\boldsymbol{x}_{[n]}$ and $\boldsymbol{x}_{[m]}$ denote *different* bitvector variables! However, we will not make use of this unless we want to reflect a semantic correspondence between the values of $\boldsymbol{x}_{[m]}$ and $\boldsymbol{x}_{[n]}$ by syntactical similarity.

Bitvector constants $\langle v_{n-1}, \ldots, v_2, v_1, v_0 \rangle \in \mathbb{B}_{[n]}$ for $n \in \mathbb{N}_+$ are denoted as binary strings $v_{n-1} \ldots v_2 v_1 v_0$ of 0's and 1's. The following abbreviations are defined:

$$0_{[n]} := \underbrace{000\ldots0}_{n} \qquad -1_{[n]} := \underbrace{111\ldots1}_{n} \qquad 1_{[n]} := \underbrace{00\ldots01}_{n-1}$$

The notation of bitvector constants resembles the notation of binary numbers, encoded with the least significant bit on the right. In general, there are several ways to interprete bitvectors as natural numbers and several ways to represent natural numbers by bitvectors. For the purpose of this thesis, we define the following conversion function which constitutes how bitvectors are mapped onto natural numbers.

Definition 3.4 (Interpretation of Bitvectors as Natural Numbers) *In order to interprete fixed-size bitvectors as natural numbers let* $\mathsf{bv2nat} : \mathbb{B}_{[*]} \longrightarrow \mathbb{N}$ *be defined such that for each* $n \in \mathbb{N}_+$*:*

$$\langle v_{n-1}, \ldots, v_1, v_0 \rangle \in \mathbb{B}_{[n]} \quad \mapsto \quad \sum_{i=0}^{n-1} 2^i \cdot v_i$$

■

The representation relation induced by Definition 3.4 is one-to-many. Interpretation of a bitvector as a natural number is well-defined, but in general there exist different bitvectors which represent the same natural number. Consider, for example, $1001 \in \mathbb{B}_{[4]}$ and $001001 \in \mathbb{B}_{[6]}$ with $\mathsf{bv2nat}(1001) = 9 = \mathsf{bv2nat}(001001)$. In such a case, the longer one of the two bitvectors is a zero-extension of the shorter one. Yet, a natural number can uniquely be represented by a bitvector, if additionally the width of the bitvector is specified. In the following, a general encoding is defined which maps not only natural numbers, but arbitrary integers to bitvectors of specified widths, thus allowing negative numbers to be represented as well.

Definition 3.5 (Encoding of Integers as Bitvectors) *Let* $\mathsf{int2bv} : \mathbb{Z} \times \mathbb{N}_+ \longrightarrow \mathbb{B}_{[*]}$ *be defined such that*

$$(d, n) \quad \mapsto \quad \langle v_{n-1}, \ldots, v_1, v_0 \rangle \in \mathbb{B}_{[n]} \quad \textit{with} \quad \begin{cases} \sum_{i=0}^{n-1} 2^i \cdot v_i \;=\; d \bmod 2^n & \textit{if} \;\; d \geq 0 \\ \sum_{i=0}^{n-1} 2^i \cdot \overline{v_i} \;=\; (\,|d| \bmod 2^n\,) - 1 & \textit{if} \;\; d < 0 \end{cases}$$

for all $d \in \mathbb{Z}$ *and* $n \in \mathbb{N}_+$*, where* $(\cdot \bmod m) : \mathbb{N} \longrightarrow \{\,0, \ldots, m-1\,\}$ *for* $m \in \mathbb{N}_+$ *with the usual semantics.* ■

Encoding of integers as natural numbers is well-defined because representation of a natural number $d \in \mathbb{N}$ as a binary number without leading zeros is unique. The value of d is mapped onto a bitvector of width $n \in \mathbb{N}_+$ by modulus computation. For non-positive integers, int2bv yields the binary complement on two.

3.2 Bitvector Functions

A *bitvector function* is a function which maps tuples, consisting of a fixed number $k \in \mathbb{N}_+$ of bitvectors, to bitvectors of a fixed width $n \in \mathbb{N}_+$. The number k is called the *arity* of the bitvector function. The domain of a k-ary bitvector function is the ordered product space of k sets of fixed-size bitvectors of specific widths $n_1, \ldots, n_k \in \mathbb{N}_+$, and the range is the set of all bitvectors of width n.

Definition 3.6 (Bitvector Functions) *Let $k \in \mathbb{N}_+$. Let $n \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. A **$k$-ary bitvector function** on bitvectors of widths $n_1, \ldots, n_k$ is a function*

$$\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]} .$$

A bitvector function is said to be of width n if the range is $\mathbb{B}_{[n]}$. ■

Bitvector functions will be denoted by upper-case calligraphic characters which take a lower index in square brackets, indicating the width of the function similar to the chosen notation of variables for fixed-size bitvectors. A bitvector function of width 1 for which additionally all argument bitvectors are also of width 1 is called a *Boolean function*.

Definition 3.7 (Boolean Functions) *Let $k \in \mathbb{N}_+$. A k-ary bitvector function*

$$\mathcal{B}_{[1]} : \underbrace{\mathbb{B}_{[1]} \times \ldots \times \mathbb{B}_{[1]}}_{k} \longrightarrow \mathbb{B}_{[1]}$$

*of width 1 on bitvectors of width 1 is called a **Boolean function**.* ■

3.3 Bitvector Operators

Bitvector functions can be used as functional specifications of high-level operations on fixed-size bitvectors. In this section, several families of bitvector functions are defined which constitute a standard set of high-level operators on fixed-size bitvectors. First, we define *concatenation* of fixed-size bitvectors.

Definition 3.8 (Concatenation) *For $m, n \in \mathbb{N}_+$ let*

$$\otimes_{m,n} : \mathbb{B}_{[m]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[m+n]}$$

be defined by

$$(\langle v_{m-1}, \ldots, v_1, v_0 \rangle, \langle w_{n-1}, \ldots, w_1, w_0 \rangle) \mapsto \langle v_{m-1}, \ldots, v_1, v_0, w_{n-1}, \ldots, w_1, w_0 \rangle$$

for all $\langle v_{m-1}, \ldots, v_1, v_0 \rangle \in \mathbb{B}_{[m]}$ and $\langle w_{n-1}, \ldots, w_1, w_0 \rangle \in \mathbb{B}_{[n]}$. ■

Concatenation computes an ordered composition of two bitvectors. Concatenation is a non-commutative operation. The result of the concatenation of two bitvectors $\boldsymbol{x}_{[8]} \in \mathbb{B}_{[8]}$ of width 8 and $\boldsymbol{y}_{[4]} \in \mathbb{B}_{[4]}$ of width 4 is illustrated in Figure 3.2.

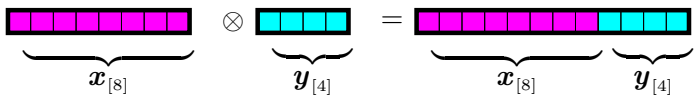


Figure 3.2: Illustration of Concatenations

We use infix notation for concatenation, and operator indices are omitted whenever they are obvious from the context, which is usually the case since bitvector variables are typed. For example, we write

$$\boldsymbol{x}_{[m]} \otimes \boldsymbol{y}_{[n]}$$

instead of

$$\otimes_{m,n}(\boldsymbol{x}_{[m]}, \boldsymbol{y}_{[n]})$$

for bitvectors $\boldsymbol{x}_{[m]} \in \mathbb{B}_{[m]}$ and $\boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. The operation which is reverse to concatenation of bitvectors is called *extraction*.

Definition 3.9 (Extraction) *For $n \in \mathbb{N}_+$ and $j, i \in \mathbb{N}$ with $0 \leq i \leq j < n$ we define*

$$[j,i]_n : \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[j-i+1]}$$

with

$$\langle v_{n-1}, \ldots, v_j, \ldots, v_i, \ldots, v_0 \rangle \quad \mapsto \quad \langle v_j, v_{j-1}, \ldots, v_{i+1}, v_i \rangle$$

for all $\langle v_{n-1}, \ldots, v_0 \rangle \in \mathbb{B}_{[n]}$. ■

Extraction selects a contiguous subrange of a bitvector, specified by a left and right index delimiter. An example of extraction operations is given in Figure 3.3 below.

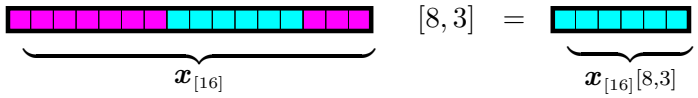


Figure 3.3: Illustration of Extractions

A contiguous subrange $\boldsymbol{x}_{[n]}[j,i]$ of a bitvector $\boldsymbol{x}_{[n]} \in \mathbb{B}_{[n]}$ is also referred to as a *chunk* of $\boldsymbol{x}_{[n]}$ (see also Section 6.2).

We use postfix notation for extraction, and indices are omitted whenever obvious from the context, i.e. we write

$$\boldsymbol{x}_{[n]}[j,i]$$

instead of

$$[j,i]_n(\boldsymbol{x}_{[n]})$$

for a bitvector $\boldsymbol{x}_{[n]}$ of width n. If both extraction delimiters i and j are equal, then extraction coincides with projection to the i^{th} bit. In that case, one delimiter is omitted and extractions $\boldsymbol{x}_{[n]}[i,i]$ are abbreviated by $\boldsymbol{x}_{[n]}[i]$.

We now address *Boolean operations* on fixed-size bitvectors. Boolean operations on bitvectors are defined as bitwise application of the respective Boolean connective on the single bits of the argument bitvectors.

Definition 3.10 (Negation) *For $n \in \mathbb{N}_+$ we define*

$$\mathsf{neg}_n : \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$$

with

$$\langle v_{n-1}, \ldots, v_1, v_0 \rangle \quad \mapsto \quad \langle \overline{v}_{n-1}, \ldots, \overline{v}_1, \overline{v}_0 \rangle$$

for all $\langle v_{n-1}, \ldots, v_0 \rangle \in \mathbb{B}_{[n]}$. ■

Negation is a unary operator which inverts each single bit of a bitvector. We use prefix notation for negation, and indices are omitted whenever obvious from the context, e.g. we write

$$\mathsf{neg}(\boldsymbol{x}_{[n]})$$

instead of

$$\mathsf{neg}_n(\boldsymbol{x}_{[n]})\,.$$

Further Boolean operations on bitvectors are defined with respect to the well-known binary Boolean connectives and, or, xor, nand, nor and xnor.

Definition 3.11 (Bitwise Boolean Connectives) *For $n \in \mathbb{N}_+$ and $\odot \in \{$*and*,* or*,* xor*,* nand*,* nor*,* xnor*$\}$ we define*

$$\odot_n : \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$$

with

$$(\,\langle v_{n-1}, \ldots, v_1, v_0 \rangle, \langle w_{n-1}, \ldots, w_1, w_0 \rangle\,) \quad \mapsto \quad \langle v_{n-1} \odot w_{n-1}, \ldots, v_1 \odot w_1, v_0 \odot w_0 \rangle$$

for all $\langle v_{n-1}, \ldots, v_1, v_0 \rangle \in \mathbb{B}_{[n]}$ and $\langle w_{n-1}, \ldots, w_1, w_0 \rangle \in \mathbb{B}_{[n]}$. ■

The Boolean connective $\odot$ is applied bitwise to each index position. The semantics of operation of $\odot \in \{\mathsf{and}, \mathsf{or}, \mathsf{xor}, \mathsf{nand}, \mathsf{nor}, \mathsf{xnor}\}$ on single bits is defined as usual. All bitwise Boolean operators on bitvectors are commutative, and we use infix notation for bitwise Boolean operations. Operator indices are omitted whenever they are obvious from the context, i.e. we write

$$\boldsymbol{x}_{[n]} \,\mathsf{and}\, \boldsymbol{y}_{[n]}$$

instead of

$$\mathsf{and}_n(\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]})$$

for bitvectors $\boldsymbol{x}_{[n]} \in \mathbb{B}_{[n]}$ and $\boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$.

The next two operators are used to model *dynamic data dependencies* (see also Chapter 8). Depending on the result of a dynamic comparison, a conditional selection of one bitvector out of two bitvectors is performed. The *if-equal-then-else* operator selects depending on the equality of its first two arguments.

Definition 3.12 (If-Equal-Then-Else) *For $m, n \in \mathbb{N}_+$ let*

$$\mathsf{ite}^{=}_{m,n} : \mathbb{B}_{[m]} \times \mathbb{B}_{[m]} \times \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$$

be defined by

$$(\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}) \;\mapsto\; \begin{cases} \boldsymbol{x}_{[n]} & \text{if } \boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]} \\ \boldsymbol{y}_{[n]} & \text{if } \boldsymbol{a}_{[m]} \neq \boldsymbol{b}_{[m]} \end{cases}$$

for all $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]} \in \mathbb{B}_{[m]}$ and $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. ■

We use prefix notation for if-equal-then-else operations, and operator indices are omitted whenever obvious from the context. Furthermore, to provide a convenient notation, we will place the equality operator within the conditional part of the ite operator, i.e. between the first two arguments. For example, we will write

$$\mathsf{ite}(\boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]})$$

instead of

$$\mathsf{ite}^{=}_{m,n}(\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]})$$

for bitvectors $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]} \in \mathbb{B}_{[m]}$ and $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. Correspondingly, we define if-then-else operators with strict comparison used in the conditional part.

Definition 3.13 (If-Less-Then-Else) *For $m, n \in \mathbb{N}_+$ let*

$$\mathsf{ite}^{<}_{m,n} : \mathbb{B}_{[m]} \times \mathbb{B}_{[m]} \times \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$$

be defined by

$$(\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}) \quad \mapsto \quad \begin{cases} \boldsymbol{x}_{[n]} & \textit{if } \mathsf{bv2nat}(\boldsymbol{a}_{[m]}) < \mathsf{bv2nat}(\boldsymbol{b}_{[m]}) \\ \boldsymbol{y}_{[n]} & \textit{if } \mathsf{bv2nat}(\boldsymbol{a}_{[m]}) \geq \mathsf{bv2nat}(\boldsymbol{b}_{[m]}) \end{cases}$$

for all $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]} \in \mathbb{B}_{[m]}$ and $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. ■

For *if-less-then-else* operators, the conditional selection depends on strict comparison of the first two operands. Bitvector values are interpreted as unsigned natural numbers. Again, prefix notation is used, and operator indices are omitted whenever they can be derived from the context. For example, we write

$$\mathsf{ite}(\boldsymbol{a}_{[m]} < \boldsymbol{b}_{[m]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]})$$

instead of

$$\mathsf{ite}^{<}_{m,n}(\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]})$$

for bitvectors $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]} \in \mathbb{B}_{[m]}$ and $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$.

Next, *arithmetic operations* on bitvectors are defined. We define bitvector arithmetics on bitvectors of width $n \in \mathbb{N}_+$ according to *ring arithmetics* in the *residue ring* $\mathbb{Z}_{2^n}$. Again, bitvector values are interpreted as unsigned natural numbers.

Definition 3.14 (Bitvector Arithmetics) *For each $n \in \mathbb{N}_+$ we define the following arithmetic operations on bitvectors of width n:*

$$\begin{array}{rcl} \oplus_n & : & \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]} \\ \ominus_n & : & \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]} \\ \circledast_n & : & \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]} \end{array}$$

with

$$\begin{array}{rcl} \oplus_n(\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}) & := & \mathsf{int2bv}(\ \mathsf{bv2nat}(\boldsymbol{x}_{[n]}) + \mathsf{bv2nat}(\boldsymbol{y}_{[n]}),\ n\ ) \\ \ominus_n(\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}) & := & \mathsf{int2bv}(\ \mathsf{bv2nat}(\boldsymbol{x}_{[n]}) - \mathsf{bv2nat}(\boldsymbol{y}_{[n]}),\ n\ ) \\ \circledast_n(\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}) & := & \mathsf{int2bv}(\ \mathsf{bv2nat}(\boldsymbol{x}_{[n]}) \cdot \mathsf{bv2nat}(\boldsymbol{y}_{[n]}),\ n\ ) \end{array}$$

for all $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. ■

The $\oplus_n$ operator performs n-bit addition, $\ominus_n$ the n-bit subtraction, and $\circledast_n$ describes n-bit multiplication. All results are again bitvectors of width n. The $\oplus_n$, $\ominus_n$ and $\circledast_n$ operators are used to describe arithmetic operations as they are performed in digital hardware.

We use infix notation for bitvector arithmetic, and indices are omitted whenever obvious from the context, e.g. we write

$$\boldsymbol{x}_{[n]} \oplus \boldsymbol{y}_{[n]}$$

instead of

$$\oplus_n(\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]})$$

for bitvectors $\boldsymbol{x}_{[n]} \in \mathbb{B}_{[n]}$ and $\boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. Note, that overflow bits of an n-bit operation can be inspected by using concatenations for a prior extension of the argument values (cf. Definition 3.18), subsequent use of $(n+1)$-bit arithmetics, and then using extractions:

$$\boldsymbol{c}_{[1]} \quad = \quad (\,(0_{[1]} \otimes \boldsymbol{x}_{[n]}) \oplus_{n+1} (0_{[1]} \otimes \boldsymbol{y}_{[n]})\,)[n]$$

The next two classes of bitvector operators are used to describe memory accesses in digital circuits. In Definition 3.9, extraction with constant range delimiters is defined, also called *low-level extraction*. In the following, operators for *high-level extractions* and for *high-level assignments* are defined, which can be used to model read and write accesses to arrays of memory cells.

Definition 3.15 (High-Level Read) *For $l, m, n \in \mathbb{N}_+$ with $l = 2^m \cdot n$ we define*

$$\mathsf{read}_{l,m,n} : \mathbb{B}_{[l]} \times \mathbb{B}_{[m]} \longrightarrow \mathbb{B}_{[n]}$$

such that for all $\boldsymbol{array}_{[l]} \in \mathbb{B}_{[l]}$ and $\boldsymbol{index}_{[m]} \in \mathbb{B}_{[m]}$

$$\mathsf{read}_{l,m,n}(\boldsymbol{array}_{[l]}, \boldsymbol{index}_{[m]}) \;:=\; \boldsymbol{array}_{[l]}[i+n-1, i]$$

with $i \;:=\; \mathsf{bv2nat}(\boldsymbol{index}_{[m]}) \cdot n.$ ■

The read-*operator* selects a specific chunk of a bitvector, specified by an index parameter which itself is a bitvector. The first operand is interpreted as an array of bitvector cells which are numbered increasingly from right to left, starting with cell 0. The extraction index is determined by the value of the second operand, interpreted as a natural number. An example is given in Figure 3.4, illustrating high-level extraction from an array of 4 cells of 4-bit width.

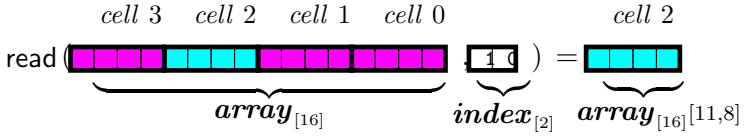


Figure 3.4: Illustration of Read Operations

For read operations, the high-level width of the array of bitvector cells, i.e. the number of cells, is constrained to be a power of 2, determined by the width of the index bitvector. Thus read operations are always well-defined and no index violations can occur.

The complementary operation is described by the write-operator, which performs a *high-level assignment* and can be used to model a write access to an array of memory cells.

Definition 3.16 (High-Level Write) *Let $l, m, n \in \mathbb{N}_+$ such that $l = 2^m \cdot n$. For all $\boldsymbol{array}_{[l]} \in \mathbb{B}_{[l]}$, $\boldsymbol{index}_{[m]} \in \mathbb{B}_{[m]}$ and $\boldsymbol{value}_{[n]} \in \mathbb{B}_{[n]}$ let*

$$\mathsf{write}_{l,m,n} : \mathbb{B}_{[l]} \times \mathbb{B}_{[m]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[l]}$$

be defined by

$$\mathsf{write}_{l,m,n}(\boldsymbol{array}_{[l]}, \boldsymbol{index}_{[m]}, \boldsymbol{value}_{[n]}) := \boldsymbol{array'}_{[l]}$$

with

$$\boldsymbol{array'}_{[l]} = \begin{cases} \boldsymbol{array}_{[l]}[l-1, n] \otimes \boldsymbol{value}_{[n]} & \text{if} \quad i = 0 \\ \boldsymbol{array}_{[l]}[l-1, i+n] \otimes \boldsymbol{value}_{[n]} \otimes \boldsymbol{array}_{[l]}[i-1, 0] & \text{if} \quad 0 < i < l-n \\ \boldsymbol{value}_{[n]} \otimes \boldsymbol{array}_{[l]}[l-n-1, 0] & \text{if} \quad i = l-n \end{cases}$$

and $i := \mathsf{bv2nat}(\boldsymbol{index}_{[m]}) \cdot n$. ■

Similar to read operations, the first argument of a write operation specifies an array of bitvector cells, and the second argument specifies the high-level cell position. The result of a write operation is the modified array of memory cells in which the specific cell which is referenced by the index bitvector is replaced by the third argument, which is a bitvector of the same width as the array cells, as it is exemplified in Figure 3.5.

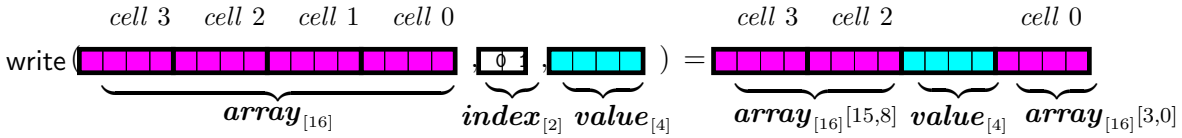


Figure 3.5: Illustration of Write Operations

We use prefix notation for read and write operations, and indices are omitted whenever obvious from the context, e.g. we write

$$\mathsf{read}(\boldsymbol{array}_{[l]}, \boldsymbol{index}_{[m]}) \qquad \text{and} \qquad \mathsf{write}(\boldsymbol{array}_{[l]}, \boldsymbol{index}_{[m]}, \boldsymbol{value}_{[n]})$$

instead of

$$\mathsf{read}_{l,m,n}(\boldsymbol{array}_{[l]}, \boldsymbol{index}_{[m]}) \qquad \text{and} \qquad \mathsf{write}_{l,m,n}(\boldsymbol{array}_{[l]}, \boldsymbol{index}_{[m]}, \boldsymbol{value}_{[n]})$$

for bitvectors $\boldsymbol{array}_{[l]} \in \mathbb{B}_{[l]}$, $\boldsymbol{index}_{[m]} \in \mathbb{B}_{[m]}$ and $\boldsymbol{value}_{[n]} \in \mathbb{B}_{[n]}$ with $l = 2^m \cdot n$. Note that the width n of the result bitvector of read operations is uniquely determined by the values of l and m.

We define three more operators which do not directly describe operations performed in digital hardware and which are mainly used in formal reasoning and the formal proofs given in Chapter 4. The first operator performs multiple concatenation of a bitvector with itself. Multiple concatenation of the same bitvector is called *repetition* and defined as follows.

Definition 3.17 (Repeat) *Let*

$$\mathsf{repeat} : \mathbb{B}_{[*]} \times \mathbb{N}_+ \longrightarrow \mathbb{B}_{[*]}$$

be defined by

$$\mathsf{repeat}(\boldsymbol{x}_{[n]}, i) \;:=\; \underbrace{\boldsymbol{x}_{[n]} \otimes \ldots \otimes \boldsymbol{x}_{[n]}}_{i}$$

for all $\boldsymbol{x}_{[n]} \in \mathbb{B}_{[*]}$ *and all* $i \in \mathbb{N}_+$. ■

For $n, i \in \mathbb{N}_+$ and $\boldsymbol{x}_{[n]} \in \mathbb{B}_{[n]}$ the repeat-operator yields the i-fold concatenation of its argument bitvector $\boldsymbol{x}_{[n]}$, as shown in Figure 3.6 below.

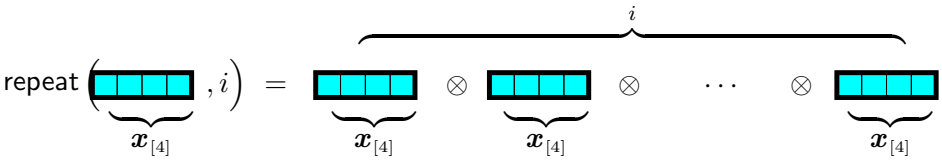


Figure 3.6: Illustration of Repeat Operations

The signExt-operator computes a *signed extension* of the value of a bitvector. A bitvector $\langle v_{m-1}, \ldots, v_0 \rangle \in \mathbb{B}_{[m]}$ of width $m \in \mathbb{N}_+$ is extended to a bitvector $\langle w_{n-1}, \ldots, w_0 \rangle \in \mathbb{B}_{[n]}$ of width $n \in \mathbb{N}_+$ with $n > m$ by copying the value of the most significant bit v_{m-1} into all bits w_i with $m \leq i \leq n-1$.

Definition 3.18 (Signed Extension) *For* $m, n \in \mathbb{N}_+$ *with* $n > m$ *we define*

$$\mathsf{signExt}_{m,n} : \mathbb{B}_{[m]} \longrightarrow \mathbb{B}_{[n]}$$

such that

$$\mathsf{signExt}_{m,n}(\boldsymbol{x}_{[m]}) \;:=\; \mathsf{repeat}(\boldsymbol{x}_{[m]}[m-1],\, n-m) \;\otimes\; \boldsymbol{x}_{[m]}$$

for all $\boldsymbol{x}_{[m]} \in \mathbb{B}_{[m]}$. ■

Signed extension of bitvectors is illustrated in Figure 3.7.

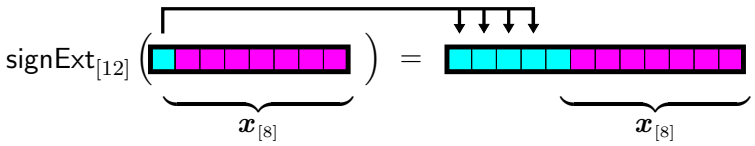


Figure 3.7: Illustration of Signed Extension

If the width m of the argument bitvector of a signed extension is obvious from the context, then we rather write

$$\mathsf{signExt}_{[n]}(\boldsymbol{x}_{[m]})$$

instead of

$$\mathsf{signExt}_{m,n}(\boldsymbol{x}_{[m]}).$$

for bitvectors $\boldsymbol{x}_{[m]} \in \mathbb{B}_{[m]}$.

We conclude this section by defining the *delta operator* which indicates equality and disequality of two bitvectors. Delta operations take two bitvectors of a width $n \in \mathbb{N}_+$ as arguments and return either the m-bit one-vector or the m-bit zero-vector depending on whether the given argument bitvectors are equal or not, where $m \in \mathbb{N}_+$ is an additional parameter of the operation.

Definition 3.19 (Delta Operator) *For $n, m \in \mathbb{N}_+$ we define*

$$\delta_{n,m} : \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[m]}$$

such that

$$\delta_{n,m}(\boldsymbol{x}_{[n]},\, \boldsymbol{y}_{[n]}) \;:=\; \begin{cases} 1_{[m]} & \text{if} \quad \boldsymbol{x}_{[n]} = \boldsymbol{y}_{[n]} \\ 0_{[m]} & \text{if} \quad \boldsymbol{x}_{[n]} \neq \boldsymbol{y}_{[n]} \end{cases}$$

for all $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. ■

If the width n of the argument bitvectors of a δ-operation is obvious from the context, then we rather write

$$\delta_{[m]}(\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]})$$

instead of

$$\delta_{n,m}(\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]})$$

for bitvectors $\boldsymbol{x}_{[n]} \in \mathbb{B}_{[n]}$ and $\boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$.

3.4 Bitwise Bitvector Functions

In this section, the notion of *bitwise bitvector functions* is defined. Bitwise bitvector functions are a specific sub-class of bitvector functions which have strictly symmetric and uniform data dependencies and a strictly symmetric and uniform data flow.

In general, bit-level data dependencies and bit-level data flow of an arbitrary bitvector function $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ with $k, n \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$ can be characterized by n Boolean bitvector functions $\mathcal{B}^0_{[1]}, \ldots, \mathcal{B}^{n-1}_{[1]}$, where for each $i \in \{0, \ldots, n-1\}$

$$\mathcal{B}^i_{[1]} : \underbrace{\mathbb{B}_{[1]} \times \ldots \times \mathbb{B}_{[1]}}_{l} \longrightarrow \mathbb{B}_{[1]}$$

is of arity $l := \sum_{j \in \{1,\ldots,k\}} n_j$, and

$$\begin{aligned}\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})[i] \quad = \quad & \mathcal{B}^i_{[1]}(\boldsymbol{x}^1_{[n_1]}[n_1-1], \boldsymbol{x}^1_{[n_1]}[n_1-2], \ldots, \boldsymbol{x}^1_{[n_1]}[1], \boldsymbol{x}^1_{[n_1]}[0], \\ & \boldsymbol{x}^2_{[n_2]}[n_2-1], \boldsymbol{x}^2_{[n_2]}[n_2-2], \ldots, \boldsymbol{x}^2_{[n_2]}[1], \boldsymbol{x}^2_{[n_2]}[0], \\ & \ldots, \\ & \boldsymbol{x}^k_{[n_k]}[n_k-1], \boldsymbol{x}^k_{[n_k]}[n_k-2], \ldots, \boldsymbol{x}^k_{[n_k]}[1], \boldsymbol{x}^k_{[n_k]}[0])\end{aligned}$$

for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^1_{[n_k]} \in \mathbb{B}_{[n_k]}$ and each $i \in \{0, \ldots, n-1\}$. In general, $\mathcal{B}^i_{[1]}$ does not necessarily have to depend truly on all its arguments $\boldsymbol{x}^i_{[n_i]}[j]$, but only on a certain subset, as illustrated in Figure 3.8 (the arrows are used to indicate functional dependencies between bits).

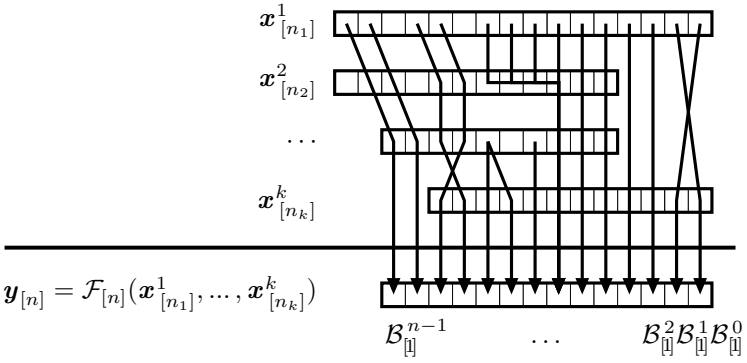


Figure 3.8: General Bit-Level Data Flow and Bit-Level Data Dependencies

A bitvector function $\mathcal{F}_{[n]}$ of width $n \in \mathbb{N}_+$ is called *bitwise* if all argument bitvectors are also of width n, that is $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$, and if $\mathcal{F}_{[n]}$ operates uniformly on all bit positions of its arguments according to the same fixed Boolean function $\mathcal{B}_{[1]}$.

Definition 3.20 (Bitwise Bitvector Functions) *Let $n \in \mathbb{N}_+$ and let $k \in \mathbb{N}_+$. Let furthermore $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitvector function of width n on bitvectors of width n. Then $\mathcal{F}_{[n]}$ is a **bitwise bitvector function** if there exists a k-ary Boolean function $\mathcal{B}_{[1]} : \mathbb{B}_{[1]} \times \ldots \times \mathbb{B}_{[1]} \longrightarrow \mathbb{B}_{[1]}$ such that for all $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ and all $i \in \{0, \ldots, n-1\}$ we have*

$$\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]})[i] \;=\; \mathcal{B}_{[1]}(\boldsymbol{x}^1_{[n]}[i], \ldots, \boldsymbol{x}^k_{[n]}[i])\,.$$

*$\mathcal{B}_{[1]}$ is then called the **characteristic Boolean function** of $\mathcal{F}_{[n]}$.* ■

The bit-level data dependencies and the bit-level data flow of bitwise bitvector functions are illustrated in Figure 3.9.

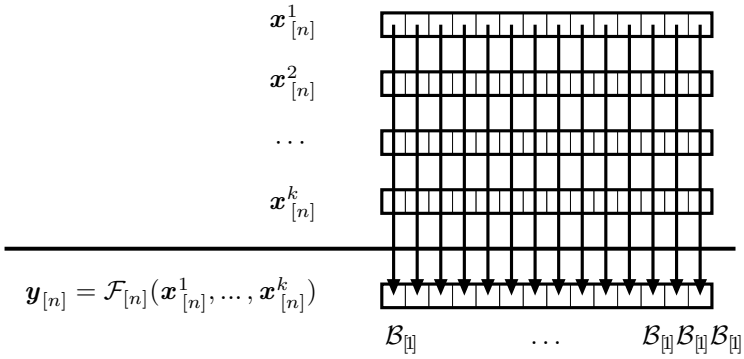


Figure 3.9: Uniform Data Flow and Uniform Data Dependencies

Note: If a bitvector function $\mathcal{F}_{[n]}$ is bitwise, then the characteristic Boolean function $\mathcal{B}_{[1]}$ of $\mathcal{F}_{[n]}$ is uniquely determined.

Example 3.21 (Bitwise Bitvector Functions) *The bitvector operators introduced in Definitions 3.10 and 3.11 are bitwise bitvector functions.* □

Example 3.22 (Bitwise Bitvector Functions) *For all $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]} \in \mathbb{B}_{[8]}$ let the bitvector function $\mathcal{F}_{[8]} : \mathbb{B}_{[8]} \times \mathbb{B}_{[8]} \times \mathbb{B}_{[8]} \longrightarrow \mathbb{B}_{[8]}$ be defined by*

$$\mathcal{F}_{[8]}(\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]}) \;:=\; (\boldsymbol{x}_{[8]} \text{ and } \boldsymbol{y}_{[8]}) \text{ or } \boldsymbol{z}_{[8]}\,,$$

and for all $\boldsymbol{a}_{[1]}, \boldsymbol{b}_{[1]}, \boldsymbol{c}_{[1]} \in \mathbb{B}_{[1]}$ let $\mathcal{B}_{[1]} : \mathbb{B}_{[1]} \times \mathbb{B}_{[1]} \times \mathbb{B}_{[1]} \longrightarrow \mathbb{B}_{[1]}$ be defined by

$$\mathcal{B}_{[1]}(\boldsymbol{a}_{[1]}, \boldsymbol{b}_{[1]}, \boldsymbol{c}_{[1]}) \;:=\; (\boldsymbol{a}_{[1]} \text{ and } \boldsymbol{b}_{[1]}) \text{ or } \boldsymbol{c}_{[1]}\,.$$

Then $\mathcal{F}_{[8]}$ is bitwise with characteristic Boolean function $\mathcal{B}_{[1]}$. □

We close this section by examining certain closure properties of bitwise bitvector functions. We show that bitwise bitvector functions are closed under negation and Boolean composition as defined in Definition 3.10 and Definition 3.11.

Corollary 3.23 (Negation of Bitwise Bitvector Functions) *Let $n, k \in \mathbb{N}_+$ and let $\mathcal{F}_{[n]}$ be a k-ary bitwise bitvector function of width n on bitvectors of width n. Let $\mathcal{G}_{[n]}$ be the k-ary bitvector function of width n on bitvectors of width n defined by*

$$\mathcal{G}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) \quad := \quad \mathsf{neg}\big(\, \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) \,\big)$$

for all $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$. Then $\mathcal{G}_{[n]}$ is a bitwise bitvector function.

Proof: Let $\mathcal{B}_{[1]}$ be the characteristic Boolean function of $\mathcal{F}_{[n]}$. Let $\mathcal{D}_{[1]}$ be defined by

$$\mathcal{D}_{[1]}(\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]}) \;:=\; \mathsf{neg}\big(\, \mathcal{B}_{[1]}(\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]}) \,\big)$$

for all $\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]} \in \mathbb{B}_{[1]}$. Let $i \in \{0, \ldots, n-1\}$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$. Then we have:

$$\begin{aligned}
\mathcal{G}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]})[i] &= \big(\, \mathsf{neg}(\, \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) \,) \,\big)[i] \\
&= \mathsf{neg}\big(\, \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]})[i] \,\big) \\
&= \mathsf{neg}\big(\, \mathcal{B}_{[1]}(\boldsymbol{x}^1_{[n]}[i], \ldots, \boldsymbol{x}^k_{[n]}[i]) \,\big) \\
&= \mathcal{D}_{[1]}(\boldsymbol{x}^1_{[n]}[i], \ldots, \boldsymbol{x}^k_{[n]}[i])
\end{aligned}$$

i.e. $\mathcal{G}_{[n]}$ is bitwise with characteristic Boolean function $\mathcal{D}_{[1]}$. ■

Corollary 3.24 (Boolean Composition of Bitwise Bitvector Functions) *Let $k \in \mathbb{N}_+$ and let $n \in \mathbb{N}_+$. Let $\mathcal{F}^1_{[n]}$ and $\mathcal{F}^2_{[n]}$ be k-ary bitwise bitvector functions of width n on bitvectors of width n. Let $\odot \in \{\mathsf{and}, \mathsf{or}, \mathsf{xor}, \mathsf{nand}, \mathsf{nor}, \mathsf{xnor}\}$ and let $\mathcal{G}_{[n]}$ be the k-ary bitvector function of width n on bitvectors of width n defined by*

$$\mathcal{G}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) \quad := \quad \mathcal{F}^1_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) \;\odot\; \mathcal{F}^2_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]})$$

for all $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$. Then $\mathcal{G}_{[n]}$ is a bitwise bitvector function.

Proof: Let $\mathcal{B}^1_{[1]}$ be the characteristic Boolean function of $\mathcal{F}^1_{[n]}$ and let $\mathcal{B}^2_{[1]}$ be the characteristic Boolean function of $\mathcal{F}^2_{[n]}$. Then $\mathcal{B}^1_{[1]} \odot \mathcal{B}^2_{[1]}$, defined by

$$(\mathcal{B}^1_{[1]} \odot \mathcal{B}^2_{[1]})(\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]}) \;:=\; \mathcal{B}^1_{[1]}(\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]}) \;\odot\; \mathcal{B}^2_{[1]}(\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]})$$

for all $\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]} \in \mathbb{B}_{[1]}$, is the characteristic Boolean bitvector function of $\mathcal{F}^1_{[n]} \odot \mathcal{F}^2_{[n]}$, i.e. the characteristic Boolean function of $\mathcal{G}_{[n]}$. ■

3.5 Corresponding Bitwise Bitvector Functions

If two k-ary bitwise bitvector functions $\mathcal{F}^1_{[n]}$ and $\mathcal{F}^2_{[m]}$ of different widths $n \in \mathbb{N}_+$ and $m \in \mathbb{N}_+$ have the same characteristic Boolean function, then $\mathcal{F}^1_{[n]}$ and $\mathcal{F}^2_{[m]}$ are called *corresponding bitwise bitvector functions*, and this correspondence is denoted by $\mathcal{F}^1_{[n]} \simeq \mathcal{F}^2_{[m]}$.

Definition 3.25 (Corresponding Bitwise Bitvector Functions) *Let $k \in \mathbb{N}_+$ and let $m, n \in \mathbb{N}_+$. Let $\mathcal{F}^1_{[n]}$ be a k-ary bitwise bitvector function of width n on bitvector of width n, and let $\mathcal{F}^2_{[m]}$ be a k-ary bitwise bitvector function of width m on bitvectors of width m. Let $\mathcal{B}^1_{[1]}$ be the characteristic Boolean function of $\mathcal{F}^1_{[n]}$, and let $\mathcal{B}^2_{[1]}$ be the characteristic Boolean function of $\mathcal{F}^2_{[m]}$. The **correspondence relation** $\simeq$ for bitwise bitvector functions is defined by*

$$\mathcal{F}^1_{[n]} \simeq \mathcal{F}^2_{[m]} \quad :\Longleftrightarrow \quad \mathcal{B}^1_{[1]} = \mathcal{B}^2_{[1]}\,.$$

*If $\mathcal{F}^1_{[n]} \simeq \mathcal{F}^2_{[m]}$, then $\mathcal{F}^1_{[n]}$ and $\mathcal{F}^2_{[m]}$ are referred to as **corresponding bitwise bitvector functions**.* ■

Note: $\simeq$ is reflexive, symmetric and transitive, i.e. $\simeq$ defines an equivalence relation on the class of all bitwise bitvector functions.

Example 3.26 (Corresponding Bitwise Bitvector Functions) *Let $\mathcal{F}_{[n]}$ be a bitwise bitvector function with characteristic Boolean function $\mathcal{B}_{[1]}$. Then $\mathcal{F}_{[n]} \simeq \mathcal{B}_{[1]}$.* □

Example 3.27 (Corresponding Bitwise Bitvector Functions) *For all $\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]} \in \mathbb{B}_{[8]}$ let $\mathcal{F}_{[8]} : \mathbb{B}_{[8]} \times \mathbb{B}_{[8]} \times \mathbb{B}_{[8]} \longrightarrow \mathbb{B}_{[8]}$ be defined by*

$$\mathcal{F}_{[8]}(\boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]}) \;:=\; \boldsymbol{x}_{[8]} \;\mathsf{and}\; (\mathsf{neg}(\boldsymbol{y}_{[8]}) \;\mathsf{or}\; \boldsymbol{z}_{[8]})\,,$$

and for all $\boldsymbol{x}_{[4]}, \boldsymbol{y}_{[4]}, \boldsymbol{z}_{[4]} \in \mathbb{B}_{[4]}$ let $\mathcal{G}_{[4]} : \mathbb{B}_{[4]} \times \mathbb{B}_{[4]} \times \mathbb{B}_{[4]} \longrightarrow \mathbb{B}_{[4]}$ be defined by

$$\mathcal{G}_{[4]}(\boldsymbol{x}_{[4]}, \boldsymbol{y}_{[4]}, \boldsymbol{z}_{[4]}) \;:=\; \boldsymbol{x}_{[4]} \;\mathsf{and}\; \mathsf{neg}(\boldsymbol{y}_{[4]} \;\mathsf{and}\; \mathsf{neg}(\boldsymbol{z}_{[4]}))\,.$$

Then $\mathcal{F}_{[8]}$ and $\mathcal{G}_{[4]}$ are bitwise bitvector functions with $\mathcal{F}_{[8]} \simeq \mathcal{G}_{[4]}$. □

Let $\mathcal{F}_{[n]}$ be a k-ary bitwise bitvector function of width $n \in \mathbb{N}_+$. Then for each $i \in \mathbb{N}_+$ there exists a uniquely determined corresponding bitwise bitvector function $\mathcal{G}_{[i]}$ with $\mathcal{F}_{[n]} \simeq \mathcal{G}_{[i]}$. The family $(\mathcal{G}_{[i]})_{i \in \mathbb{N}_+}$ is then called a *family of corresponding bitwise bitvector functions*.

Definition 3.28 (Family of Corresponding Bitwise Bitvector Functions) *Let $k \in \mathbb{N}_+$ and for each $i \in \mathbb{N}_+$ let $\mathcal{F}^i_{[i]}$ be a k-ary bitwise bitvector function of width i on bitvectors of width i. Then $\mathcal{F}^1_{[1]}, \mathcal{F}^2_{[2]}, \mathcal{F}^3_{[3]}, \ldots$ are called a **family of corresponding bitwise bitvector functions** if there exists a characteristic k-ary Boolean function $\mathcal{B}_{[1]}$ such that for all $i \in \mathbb{N}_+$ we have $\mathcal{F}^i_{[i]} \simeq \mathcal{B}_{[1]}$.* ■

If $\mathcal{F}^1_{[1]}, \mathcal{F}^2_{[2]}, \mathcal{F}^3_{[3]}, \ldots$ is a family of corresponding bitwise bitvector functions, then the common characteristic Boolean function is uniquely determined, and furthermore we have $\mathcal{F}^i_{[i]} \simeq \mathcal{F}^j_{[j]}$ for all $i, j \in \mathbb{N}_+$.

For convenience, throughout this thesis we agree upon the following notation for families of corresponding bitwise bitvector functions: whenever we use the same identifier for two bitwise bitvector functions of different widths, then both bitwise functions belong to the same family of corresponding bitwise bitvector functions, i.e. if a bitwise bitvector function $\mathcal{F}_{[n]}$ of width $n \in \mathbb{N}_+$ is given, then $\mathcal{F}_{[m]}$ denotes the corresponding bitwise bitvector function with $\mathcal{F}_{[n]} \simeq \mathcal{F}_{[m]}$.

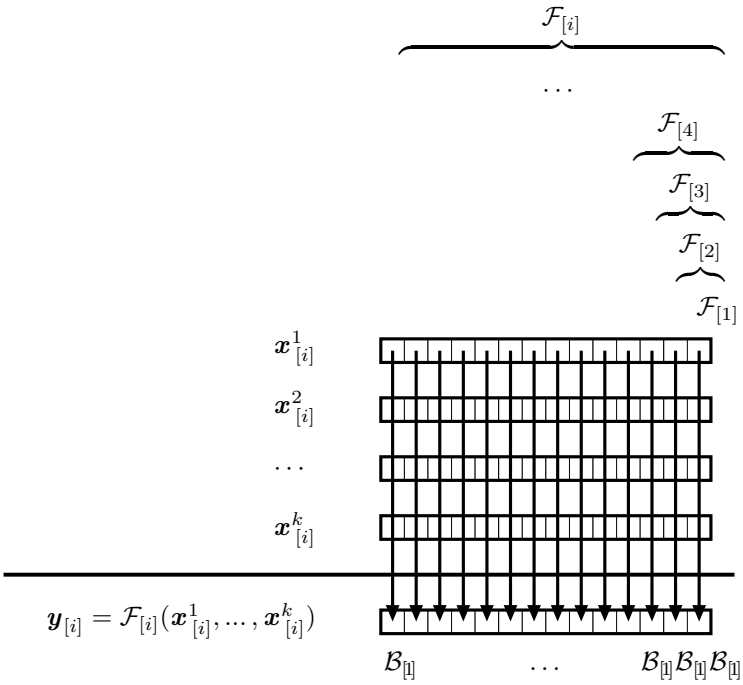


Figure 3.10: Family of Corresponding Bitwise Bitvector Functions $(\mathcal{F}_{[i]})_{i \in \mathbb{N}_+}$

Figure 3.10 illustrates the concept of a family of corresponding bitwise bitvector functions and exemplifies the chosen form of notation. Figure 3.10 can also be considered as an illustration of the following example.

Example 3.29 (Corresponding Bitwise Bitvector Functions) *For each* $i \in \mathbb{N}_+$ *let*

$$\mathcal{F}_{[i]} : \underbrace{\mathbb{B}_{[i]} \times \ldots \times \mathbb{B}_{[i]}}_{k} \longrightarrow \mathbb{B}_{[i]}$$

be defined by

$$\mathcal{F}_{[i]}(\boldsymbol{x}^1_{[i]}, \ldots, \boldsymbol{x}^k_{[i]}) \;:=\; \boldsymbol{x}^1_{[i]} \text{ or } \ldots \text{ or } \boldsymbol{x}^k_{[i]}$$

for all $\boldsymbol{x}^1_{[i]}, \ldots, \boldsymbol{x}^k_{[i]} \in \mathbb{B}_{[i]}$. *Let* $\mathcal{B}_{[1]} := \mathcal{F}_{[1]}$. *Then* $(\mathcal{F}_{[i]})_{i \in \mathbb{N}_+}$ *is a family of corresponding bitwise bitvector functions with characteristic Boolean function* $\mathcal{B}_{[1]}$. □

Chapter 4

Reducing Bitvector Satisfiability Problems

In this chapter, the *satisfiability problem* **BvSAT** *for bitvector functions* is introduced, which is an adaptation of the well known SAT problem from the Boolean domain to the bitvector domain. **BvSAT** addresses the question whether or not there exists a zero for a given bitvector function, i.e. if a given bitvector function can evaluate to a zero-vector. We investigate satisfiability of **BvSAT** problems for bitwise bitvector functions and characterize the set of satisfying solutions of bitwise instances of **BvSAT**.

Further on, we introduce an augmented version of the **BvSAT** problem, extending the original problem by a set of disequalities which imposes additional constraints on satisfying solutions. Satisfiability of the extended problem is examined for bitwise bitvector functions, and a one-to-one abstraction is presented which allows to reduce satisfiability of given bitwise instances of **BvSAT** to satisfiability of corresponding instances over smaller bitvector domains by shrinking the widths of the bitvectors. We show how satisfying solutions of the original problems can be obtained from solutions of the smaller abstract problem instances, and vice versa. Moreover, we prove that for bitwise bitvector functions the proposed abstraction yields the optimum amount of reduction, i.e. the minimum bitvector width for which satisfiability of the corresponding instances is preserved in a one-to-one fashion (for a summary see also [Joh01c]).

4.1 Preliminaries

Let $n \in \mathbb{N}_+$. According to Definition 3.2, a bitvector $\boldsymbol{x}_{[n]} \in \mathbb{B}_{[n]}$ of width n is an ordered array $\langle x_{n-1}, \ldots, x_0 \rangle$ of n bits $x_0, \ldots, x_{n-1} \in \mathbb{B}$. In the following, we will frequently consider *arrays of bitvectors*, i.e. arrays where the array elements are bitvectors. Therefore, the following notion and notation is defined:

Definition 4.1 (Arrays of Bitvectors) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. A **$k$-element array of bitvectors** is a k-tuple $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle$ with $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$.* ■

In order to distinguish between arrays of bits (i.e. bitvectors) and arrays of bitvectors, arrays of bitvectors are indicated by bold angled brackets. In case that all bitvectors which are grouped inside an array are of the same width $n \in \mathbb{N}_+$, then such an array $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ is called a *regular* array of bitvectors of width n. A regular array of bitvectors can be considered as a two-dimensional array of bits, and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}$ can be regarded as the rows of a $k \times n$ matrix, where k is the number of bitvectors and n is the width of each bitvector.

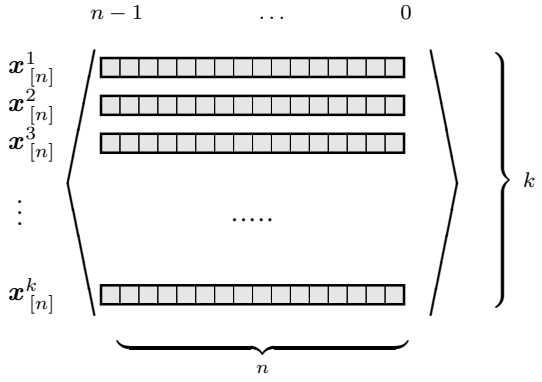


Figure 4.1: A Regular Array $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ of Bitvectors of Width n

Two operators on regular arrays of bitvectors are defined, which are *extraction* and *concatenation*. Extraction and concatenation of regular arrays of bitvectors operate component-wise on the array elements, and the result of both operations again is a regular array of bitvectors. Concatenation is defined as follows:

Definition 4.2 (Concatenation of Arrays of Bitvectors) *For $k \in \mathbb{N}_+$ and $m, n \in \mathbb{N}_+$ let*

$$\otimes_{k,m,n} \;:\; \underbrace{\mathbb{B}_{[m]} \times \ldots \times \mathbb{B}_{[m]}}_{k} \;\times\; \underbrace{\mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]}}_{k} \;\longrightarrow\; \underbrace{\mathbb{B}_{[m+n]} \times \ldots \times \mathbb{B}_{[m+n]}}_{k}$$

be defined by

$$\otimes_{k,m,n}\big(\, \langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle, \langle \boldsymbol{y}^1_{[n]}, \ldots, \boldsymbol{y}^k_{[n]} \rangle \,\big) \;:=\; \langle \boldsymbol{x}^1_{[m]} \otimes \boldsymbol{y}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[m]} \otimes \boldsymbol{y}^k_{[n]} \rangle$$

for all $\boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \in \mathbb{B}_{[m]}$ and $\boldsymbol{y}^1_{[n]}, \ldots, \boldsymbol{y}^k_{[n]} \in \mathbb{B}_{[n]}$. ■

Operator indices are omitted whenever they are obvious from the context, and, for convenience, infix notation is used for concatenation of arrays of bitvectors, i.e. we write

$$\langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle \;\otimes\; \langle \boldsymbol{y}^1_{[n]}, \ldots, \boldsymbol{y}^k_{[n]} \rangle$$

instead of

$$\otimes_{k,m,n}\big(\, \langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle, \langle \boldsymbol{y}^1_{[n]}, \ldots, \boldsymbol{y}^k_{[n]} \rangle \,\big)$$

Concatenation of regular arrays of bitvectors is illustrated in Figure 4.2 below.

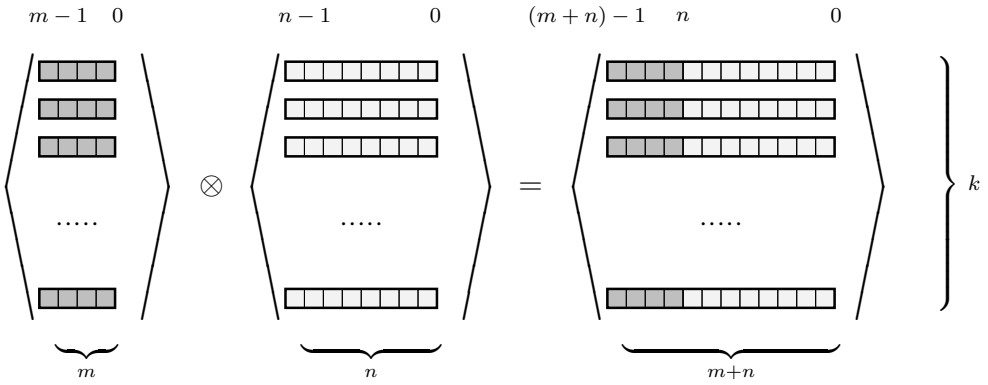


Figure 4.2: Concatenation of Arrays of Bitvectors

We now address extraction operations for regular arrays of bitvectors, which are defined as follows:

Definition 4.3 (Extraction from Arrays of Bitvectors) *For $k, n \in \mathbb{N}_+$ and $i, j \in \mathbb{N}$ with $0 \leq i \leq j < n$ we define*

$$[i,j]_{k,n} \;:\; \underbrace{\mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]}}_{k} \longrightarrow \underbrace{\mathbb{B}_{[j-i+1]} \times \ldots \times \mathbb{B}_{[j-i+1]}}_{k}$$

such that for all $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ we have:

$$[i,j]_{k,n}\big(\, \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle \,\big) \quad := \quad \langle \boldsymbol{x}^1_{[n]}[j,i], \ldots, \boldsymbol{x}^k_{[n]}[j,i] \rangle \qquad \blacksquare$$

Extraction operations select a $k \times (j - i + 1)$ sub-matrix of a given array of bitvectors, as exemplified in Figure 4.3 below.

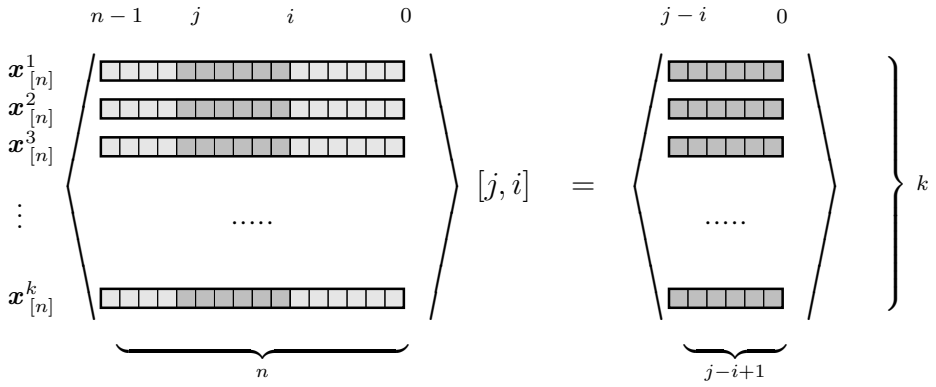


Figure 4.3: Extraction $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle [j,i]$ for Regular Arrays of Bitvectors

Postfix notation is used for extraction operations, and operator indices will be suppressed when obvious. If extraction delimiters i and j are equal, then extraction $[i, j]$ is abbreviated by $[i]$. For example, we write

$$\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle [i]$$

instead of

$$[i,i]_{k,n}\big(\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle \big)$$

Extraction and concatenation operations on arrays of bitvectors are essentially used to select specific columns of an array, and then to compose the bits of these columns to new bitvectors, and to arrange them in an array again. This concept is illustrated in Figure 4.2.

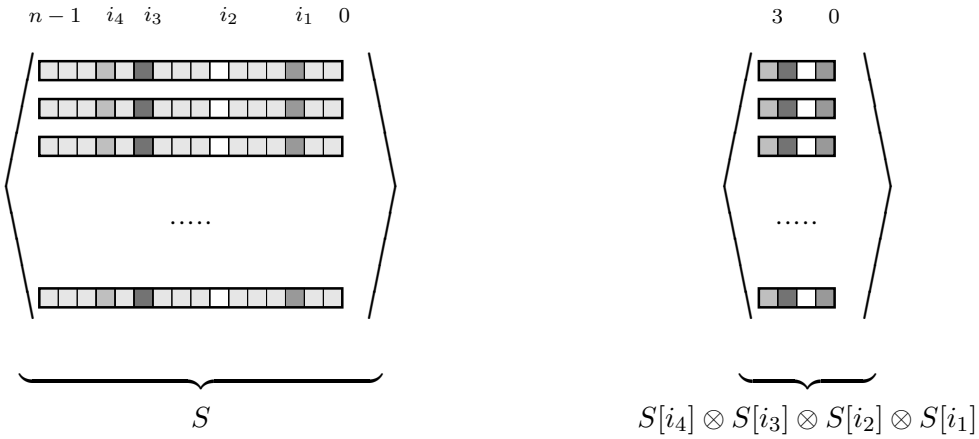


Figure 4.4: Column-Extraction and Concatenation

4.2 The Satisfiability Problem BvSAT for Bitvector Functions

In this section, the *satisfiability problem* **BvSAT** for bitvector functions is introduced. Given a k-ary bitvector function $\mathcal{F}_{[n]}$ of width n on bitvectors of widths $n_1, \ldots, n_k$, the **BvSAT** problem asks whether or not $\mathcal{F}_{[n]}$ has a zero, i.e. if the equation $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[n]}$ is satisfiable.

Definition 4.4 (The Satisfiability Problem BvSAT) *Let $k \in \mathbb{N}_+$ and let $n \in \mathbb{N}_+$. Let $n_1, \ldots, n_k \in \mathbb{N}_+$ and $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitvector function of width n on bitvectors of widths $n_1, \ldots, n_k$. Then* **BvSAT**$(\mathcal{F}_{[n]})$ *denotes the problem whether or not there exist bitvectors $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ such that $\mathcal{F}_{[n]}$ evaluates to the n-bit zero-vector, i.e.*

BvSAT$(\mathcal{F}_{[n]})$ *is satisfiable* $:\Longleftrightarrow$

$$\exists\ \boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}\ :\ \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[n]} \qquad \blacksquare$$

The satisfiability problem **BvSAT** is decidable due to finiteness of the bitvector domains. In order to be able to reason about complexity of **BvSAT**, a formal representation of bitvector functions needs to be defined. This is done in Chapter 5, where a term language for bitvector formulae is introduced. In the following, whenever complexity of **BvSAT** is considered, we implicitly assume that bitvector functions are represented in this way. In particular, this language includes arbitrary Boolean formulae. Thus, **BvSAT** can be considered as a generalization of the known SAT problem from the Boolean domain to the bitvector domain. **BvSAT** is NP-complete, which is shown by reducing SAT to **BvSAT** by mapping an instance β of SAT onto $\mathbf{BvSAT}(\mathcal{B}_{[1]})$ where $\mathcal{B}_{[1]}$ is the Boolean function on bitvectors of width 1 which corresponds to the negation of the Boolean formula β, i.e. we have SAT $\leq$ **BvSAT**.

Proposition 4.5 (NP-Completeness) BvSAT *is an* NP*-complete problem.* ■

If $\mathcal{F}_{[n]}$ is a bitvector function and $\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}$ are bitvectors with $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[n]}$, then the bitvector array $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle$ is called a *satisfying solution* of $\mathbf{BvSAT}(\mathcal{F}_{[n]})$. We define the following notation for the set of all satisfying solutions of $\mathbf{BvSAT}(\mathcal{F}_{[n]})$.

Definition 4.6 (Satisfying Solutions of BvSAT) *Let $k, n \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitvector function of width n on bitvectors of widths $n_1, \ldots, n_k$. Then let*

$$\mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[n]}) \;:=\; \{\, \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \mid \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[n]} \,\}$$

denote the set of ***satisfying solutions*** *of* $\mathbf{BvSAT}(\mathcal{F}_{[n]})$. ■

If $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[n]})$ then we also say that $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle$ *satisfies* $\mathbf{BvSAT}(\mathcal{F}_{[n]})$.

4.3 BvSAT and Bitwise Bitvector Functions

In the following, satisfiability of **BvSAT** for bitwise bitvector functions is investigated. Let $\mathcal{F}_{[n]}$ be a bitwise bitvector function with characteristic Boolean function $\mathcal{B}_{[1]}$. Then $\mathbf{BvSAT}(\mathcal{F}_{[n]})$ is called a *bitwise* **BvSAT** problem of width n. According to Definition 3.20 and Definition 4.4, satisfying solutions of bitwise **BvSAT** problems can be characterized in the following way:

Lemma 4.7 (Satisfying Solutions of Bitwise BvSAT Problems) *Let $k \in \mathbb{N}_+$ and let $n \in \mathbb{N}_+$. Let $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitwise bitvector function on bitvectors of width n with characteristic Boolean function $\mathcal{B}_{[1]} : \mathbb{B}_{[1]} \times \ldots \times \mathbb{B}_{[1]} \longrightarrow \mathbb{B}_{[1]}$. Then the following characterization*

$$\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle \in \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[n]}) \iff \forall\, i \in \{0, \ldots, n-1\} : \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle[i] \in \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{B}_{[1]})$$

holds for all $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$.

Proof: $\Longrightarrow$ Let $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle \in \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[n]})$ be a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[n]})$, and let $i \in \{0, \ldots, n-1\}$. Then $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) = \mathsf{0}_{[n]}$, and as $\mathcal{F}_{[n]}$ is a bitwise bitvector function, we conclude:

$$\begin{aligned} \mathcal{B}_{[1]}(\boldsymbol{x}^1_{[n]}[i], \ldots, \boldsymbol{x}^k_{[n]}[i]) &= \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]})[i] \\ &= \mathsf{0}_{[n]}[i] \\ &= \mathsf{0}_{[1]}\,, \end{aligned}$$

i.e. $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle[i] = \langle \boldsymbol{x}^1_{[n]}[i], \ldots, \boldsymbol{x}^k_{[n]}[i] \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{B}_{[1]})$.

$\Longleftarrow$ Let $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that, for all $i \in \{0, \ldots, n-1\}$, $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle[i]$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{B}_{[1]})$. Then we have

$$\begin{aligned} \mathsf{0}_{[1]} &= \mathcal{B}_{[1]}(\boldsymbol{x}^1_{[n]}[i], \ldots, \boldsymbol{x}^k_{[n]}[i]) \\ &= \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]})[i] \end{aligned}$$

for all $i \in \{0, \ldots, n-1\}$, i.e. $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) = \mathsf{0}_{[n]}$, and thus $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[n]})$. ■

As a consequence of Lemma 4.7, satisfiability of a bitwise problem $\mathbf{BvSAT}(\mathcal{F}_{[n]})$ solely depends on satisfiability of $\mathbf{BvSAT}(\mathcal{B}_{[1]})$ where $\mathcal{B}_{[1]}$ denotes the characteristic Boolean function of $\mathcal{F}_{[n]}$.

Theorem 4.8 (BvSAT and Bitwise Bitvector Functions) *Let $k \in \mathbb{N}_+$ and let $n \in \mathbb{N}_+$. Let $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitwise bitvector function on bitvectors of width n with characteristic Boolean function $\mathcal{B}_{[1]} : \mathbb{B}_{[1]} \times \ldots \times \mathbb{B}_{[1]} \longrightarrow \mathbb{B}_{[1]}$. Then the following holds:*

$$\mathbf{BvSAT}(\mathcal{F}_{[n]}) \text{ is satisfiable} \quad \Longleftrightarrow \quad \mathbf{BvSAT}(\mathcal{B}_{[1]}) \text{ is satisfiable}$$

Proof: $\Longrightarrow$ Let $\mathbf{BvSAT}(\mathcal{F}_{[n]})$ be satisfiable and let $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ be a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[n]})$. According to Lemma 4.7 then $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle[0]$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{B}_{[1]})$.

$\Longleftarrow$ Let $\mathbf{BvSAT}(\mathcal{B}_{[1]})$ be satisfiable, and let $\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]} \in \mathbb{B}_{[1]}$ such that $\langle \boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]} \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{B}_{[1]})$. Then for all $i \in \{0, \ldots, n-1\}$ we have

$$\langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[1]}), \ldots, \mathsf{signExt}_{[n]}(\boldsymbol{x}^k_{[1]}) \rangle[i] \;=\; \langle \boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]} \rangle\,,$$

i.e. for all $i \in \{0, \ldots, n-1\}$ we have

$$\langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[1]}), \ldots, \mathsf{signExt}_{[n]}(\boldsymbol{x}^k_{[1]}) \rangle[i] \;\in\; \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{B}_{[1]})\,,$$

and according to Lemma 4.7 then $\langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[1]}), \ldots, \mathsf{signExt}_{[n]}(\boldsymbol{x}^k_{[1]}) \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[n]})$. ■

Theorem 4.8 yields that satisfiability of a bitwise **BvSAT** problem of arbitrary width $n \in \mathbb{N}_+$ can always be related to satisfiability of a corresponding **BvSAT** problem of width 1. The following theorem is a generalization of Lemma 4.7 and Theorem 4.8, and provides another characterization of sets of satisfying solutions of corresponding bitwise **BvSAT** problems.

Theorem 4.9 (Satisfying Solutions of Corresponding BvSAT Problems) *Let $k \in \mathbb{N}_+$ and $(\mathcal{F}_{[i]})_{i \in \mathbb{N}_+}$ be a family of corresponding k-ary bitwise bitvector functions. Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ is a satisfying solution of* **BvSAT**$(\mathcal{F}_{[n]})$. *Let $m \in \mathbb{N}_+$ and let $i_0, \ldots, i_{m-1} \in \{0, \ldots, n-1\}$. Then the k-element array of bitvectors of width m which is composed as*

$$\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle[i_0] \;\otimes\; \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle[i_1] \;\otimes\; \ldots \;\otimes\; \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle[i_{m-1}]$$

is a satisfying solution of **BvSAT**$(\mathcal{F}_{[m]})$.

Proof: Let $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \ldots \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitwise bitvector function on bitvectors of width n with characteristic Boolean function $\mathcal{B}_{[1]}$. Let $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ be a satisfying solution of **BvSAT**$(\mathcal{F}_{[n]})$. Let $\mathcal{F}_{[m]}$ be the corresponding k-ary bitwise bitvector function of width m with $\mathcal{F}_{[n]} \simeq \mathcal{F}_{[m]}$. Let $i_0, \ldots, i_{m-1} \in \{0, \ldots, n-1\}$ and let $\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^k_{[m]} \in \mathbb{B}_{[m]}$ with

$$\boldsymbol{y}^l_{[m]}[j] \;:=\; \boldsymbol{x}^l_{[n]}[i_j]$$

for all $1 \leq l \leq k$ and $0 \leq j < m$. Then we have

$$\langle \boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^k_{[m]} \rangle \;=\; \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle[i_0] \;\otimes\; \ldots \;\otimes\; \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle[i_{m-1}]\,.$$

Because $\mathcal{F}_{[m]} \simeq \mathcal{F}_{[n]}$, for all $j \in \{0, \ldots, m-1\}$ the following holds:

$$\begin{aligned}
\mathcal{F}_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^k_{[m]})[j] &= \mathcal{B}_{[1]}(\boldsymbol{y}^1_{[m]}[j], \ldots, \boldsymbol{y}^k_{[m]}[j]) \\
&= \mathcal{B}_{[1]}(\boldsymbol{x}^1_{[n]}[i_j], \ldots, \boldsymbol{x}^k_{[n]}[i_j]) \\
&= \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]})[i_j] \\
&= \mathsf{0}_{[n]}[i_j] \\
&= \mathsf{0}_{[1]}\,,
\end{aligned}$$

and thus $\mathcal{F}_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^k_{[m]}) = \mathsf{0}_{[m]}$. Consequently, $\langle \boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^k_{[m]} \rangle$ is a satisfying solution of **BvSAT**$(\mathcal{F}_{[m]})$. ■

Theorem 4.9 states that satisfying solutions of corresponding bitwise **BvSAT** problems can be obtained from each other by arbitrary composition of extracted columns (cf. Figure 4.2 in Section 4.1).

Note: Theorem 4.9 in particular yields that, given a satisfying solution $S = \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ of a bitwise problem **BvSAT**$(\mathcal{F}_{[n]})$, we can obtain further satisfying solutions of **BvSAT**$(\mathcal{F}_{[n]})$ by concatenation of n arbitrarily chosen columns of S.

An easy way to obtain a satisfying solution of a corresponding bitwise **BvSAT** problem of larger width, which adheres to this principle, is a signed extension of an existing solution of the smaller problem.

Corollary 4.10 (Bitwise BvSAT Problems and Signed Extension) *Let $k \in \mathbb{N}_+$ and let $(\mathcal{F}_{[i]})_{i\in\mathbb{N}_+}$ be a family of corresponding k-ary bitwise bitvector functions. Let $m, n \in \mathbb{N}_+$ with $n \geq m$. Then the following holds*

$$\langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle \in \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[m]}) \iff \langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[m]}), \ldots, \mathsf{signExt}_{[n]}(\boldsymbol{x}^k_{[m]}) \rangle \in \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[n]})$$

for all $\boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \in \mathbb{B}_{[m]}$.

Proof: Correctness of Corollary 4.10 follows immediately from Theorem 4.9. ■

4.4 Disequality Constraints

In this section, the notion of *disequality constraints for bitvectors* is introduced. Disequality constraints are constraints which are imposed on the valuations of a fixed and ordered number of bitvector variables of the same width. A disequality constraint constrains the values of two specific bitvector variables to be different.

Definition 4.11 (Disequality Constraints) *Let $k \in \mathbb{N}_+$. A **disequality constraint** e **for** k **bitvectors** is a two-element subset*

$$e \subseteq \{1, \ldots, k\}, \quad |e| = 2$$

of indices ranging between 1 and k, i.e. $e = \{ i, j \}$ with $i, j \in \{1, \ldots, k\}$ and $i \neq j$. ■

A disequality constraint e for k bitvectors is either satisfied or unsatisfied by the values of an ordered selection $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ of k bitvectors with $n \in \mathbb{N}_+$. Ordered selections of bitvectors are represented by arrays of bitvectors, and we define:

Definition 4.12 (Satisfiability of Disequality Constraints) *Let $k \in \mathbb{N}_+$ and let e be a disequality constraint for k bitvectors, $e = \{i, j\}$ with $i, j \in \{1, \ldots, k\}$ and $i \neq j$. Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ be bitvectors of width n. Then we define that*

$$\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle \text{ satisfies } e \quad :\iff \quad \boldsymbol{x}^i_{[n]} \neq \boldsymbol{x}^j_{[n]}$$ ■

In order to denote that more than one disequality constraint is imposed on an ordered selection of bitvectors, the notion of *sets of disequality constraints for bitvectors* is introduced.

Definition 4.13 (Sets of Disequality Constraints) *Let $k \in \mathbb{N}_+$. A **set of disequality constraints** for k bitvectors is a set $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$.* ■

Satisfiability of a set I of disequality constraints is defined as satisfiability of the conjunction of all disequality constraints of I.

Definition 4.14 (Satisfiability of Sets of Disequality Constraints) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ be a set of disequality constraints for k bitvectors. Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$. Then we say that*

$$\left\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \right\rangle \text{ satisfies } I \quad :\Longleftrightarrow \quad \forall\, e \in I \;:\; \left\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \right\rangle \text{ satisfies } e$$ ■

Satisfiability of disequality constraints for bitvectors can be characterized in the following way: two bitvectors $\boldsymbol{x}_{[n]} = \langle x_{n-1}, \ldots, x_0 \rangle \in \mathbb{B}_{[n]}$ and $\boldsymbol{y}_{[n]} = \langle y_{n-1}, \ldots, y_0 \rangle \in \mathbb{B}_{[n]}$ of width $n \in \mathbb{N}_+$ are different if and only if there exists an index position $i \in \{0, \ldots, n-1\}$ such that the respective corresponding bits x_i and y_i of $\boldsymbol{x}_{[n]}$ and $\boldsymbol{y}_{[n]}$ are different, i.e. disequality of two bitvectors is always determined by disequality of two bits occurring at the same index position within the two bitvectors.

Proposition 4.15 (Disequality of Bitvectors) *Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$ be bitvectors of width n. Then the following holds:*

$$\boldsymbol{x}_{[n]} \neq \boldsymbol{y}_{[n]} \quad \Longleftrightarrow \quad \exists\, i \in \{0, \ldots, n-1\} \;:\; \boldsymbol{x}_{[n]}[i] \neq \boldsymbol{y}_{[n]}[i]$$ ■

Because the bitvalue domain is a two-valued domain, Proposition 4.15 can also be stated in the following way:

Proposition 4.16 (Disequality of Bitvectors) *Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$ be bitvectors of width n. Then the following holds:*

$$\boldsymbol{x}_{[n]} \neq \boldsymbol{y}_{[n]} \quad \Longleftrightarrow \quad \exists\, i \in \{0, \ldots, n-1\} \;:\; \boldsymbol{x}_{[n]}[i] = \mathsf{neg}(\boldsymbol{y}_{[n]}[i])$$ ■

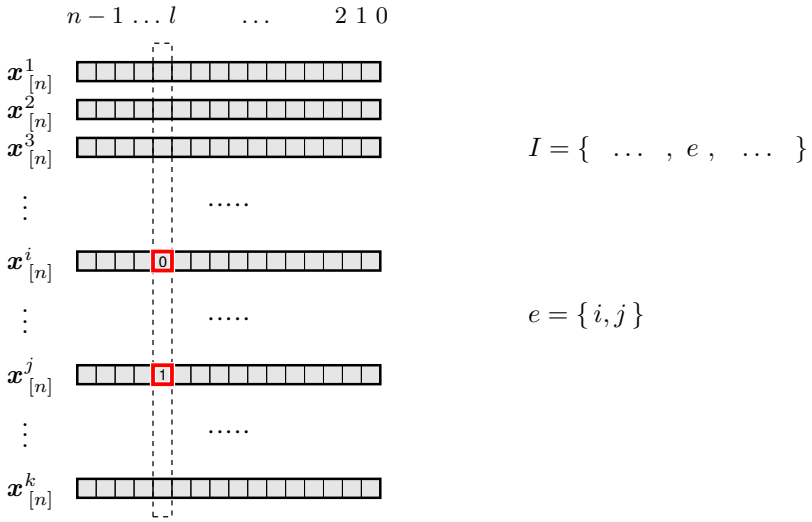
Propositions 4.15 and 4.16 yield that satisfiability of a set I of disequality constraints can be characterized in the Boolean domain by reasoning about index positions and single bits of the bitvectors.

Lemma 4.17 (Satisfiability of Sets of Disequality Constraints) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ be a set of disequality constraints for k bitvectors. Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$. Let $S := \left\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \right\rangle$. Then the following holds:*

$$S \text{ satisfies } I \quad \Longleftrightarrow \quad \forall\, e \in I \;\; \exists\, l \in \{0, \ldots, n-1\} \;:\; S[l] \text{ satisfies } e$$

Proof: Let $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ be bitvectors of width n, and let $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$.

$\Longrightarrow$ If S satisfies I, then, according to Proposition 4.15, for each disequality $e = \{i, j\} \in I$ there exists an index position l such that the l^{th} bits of the corresponding bitvectors $\boldsymbol{x}^i_{[n]}$ and $\boldsymbol{x}^j_{[n]}$ differ, as illustrated below.



Then the l^{th} column $S[l] = \langle \boldsymbol{x}^1_{[n]}[l], \ldots, \boldsymbol{x}^k_{[n]}[l] \rangle$ of $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ satisfies e in terms of Definition 4.12.

$\Longleftarrow$ Follows from Definition 4.14 and Proposition 4.15. ■

Lemma 4.17 furthermore yields that satisfiability of a set of disequality constraints is preserved by signed extension of bitvectors.

Corollary 4.18 (Disequality Constraints and Signed Extension) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{ e \subseteq \{1, \ldots, k\} \mid |e| = 2 \}$ be a set of disequality constraints for k bitvectors. Let $m \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \in \mathbb{B}_{[m]}$. Let $n \in \mathbb{N}_+$ with $n \geq m$. Then the following holds:*

$$\langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle \text{ satisfies } I \iff \langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[m]}), \ldots, \mathsf{signExt}_{[n]}(\boldsymbol{x}^k_{[m]}) \rangle \text{ satisfies } I$$

Proof: $\Longrightarrow$ Let $\boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \in \mathbb{B}_{[m]}$ such that $\langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle$ satisfies I. According to Lemma 4.17 then for all $e \in I$ there exists $l \in \{0, \ldots, m-1\}$ such that $\langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle[l]$ satisfies e. As for all $i \in \{1, \ldots, k\}$ and all $j \in \{0, \ldots, m-1\}$ we have

$$\big(\mathsf{signExt}_{[n]}(\boldsymbol{x}^i_{[m]}) \big)[j] \quad = \quad \boldsymbol{x}^i_{[m]}[j] \,,$$

this is true for $\langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[m]}), \ldots, \mathsf{signExt}_{[n]}(\boldsymbol{x}^k_{[m]}) \rangle$, too, and Lemma 4.17 in turn yields that $\langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[m]}), \ldots, \mathsf{signExt}_{[n]}(\boldsymbol{x}^k_{[m]}) \rangle$ is a satisfying solution of I.

$\Longleftarrow$ Let $\boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \in \mathbb{B}_{[m]}$ such that $\langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[m]}), \ldots, \mathsf{signExt}_{[n]}(\boldsymbol{x}^k_{[m]}) \rangle$ satisfies I. As for all $i \in \{1, \ldots, k\}$ and all $j \in \{m, \ldots, n-1\}$ we have

$$(\mathsf{signExt}_{[n]}(\boldsymbol{x}^i_{[m]}))[j] \quad = \quad \boldsymbol{x}^i_{[m]}[m-1]\,,$$

Lemma 4.17 yields that for all $e \in I$ there exists $l \in \{0, \ldots, m-1\}$ such that $\langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle[l]$ satisfies e. Thus $\langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle$ satisfies I. ■

4.5 The Enhanced Satisfiability Problem $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$

In this section, the satisfiability problem for bitvector functions is generalized and the *enhanced* **BvSAT** *problem* is introduced. The original **BvSAT** problem, which is presented in Definition 4.4, is extended by a second parameter I which specifies a set of disequalities which impose additional constraints on the bitvectors of a satisfying solution. I requires that the values of specific bitvectors of a satisfying solution must be different.

Definition 4.19 (The Enhanced Satisfiability Problem BvSAT) *Let $k, n \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitvector function of width n. Let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ with $n_i = n_j$ for all $\{i, j\} \in I$. Then $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$ denotes the problem whether there exist $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$, such that $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[n]})$ and $\boldsymbol{x}^i_{[n_i]} \neq \boldsymbol{x}^j_{[n_j]}$ for all $\{i, j\} \in I$, that is:*

$$\begin{array}{c} \mathbf{BvSAT}(\mathcal{F}_{[n]}, I) \\ \textit{is satisfiable} \end{array} \quad :\Longleftrightarrow \quad \begin{array}{l} \exists\ \boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]} : \\ \quad \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = \mathsf{0}_{[n]} \ \wedge\ \forall\ \{i, j\} \in I \ :\ \boldsymbol{x}^i_{[n_i]} \neq \boldsymbol{x}^j_{[n_j]} \end{array}$$ ■

Let $\mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[n]}, I)$ denote the set of satisfying solutions of $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$. The original **BvSAT** problem is a special case of the enhanced **BvSAT** problem where the set of disequality constraints is chosen as $I = \emptyset$. The enhanced **BvSAT** problem is in NP since **BvSAT** is and checking whether or not a given set of bitvector values satisfies a set of disequality constraints can be done in time linear to the overall number of bits. As a convention, from now on, whenever we write **BvSAT**, we will always refer to the enhanced satisfiability problem as introduced in Definition 4.19. In Proposition 4.5 we have seen that $\mathbf{BvSAT}(\cdot, \emptyset)$ is NP-complete. Thus we can immediately conclude NP-completeness of the enhanced **BvSAT** problem (assuming the same kind of representation for bitvector functions).

Proposition 4.20 (NP-Completeness) *Enhanced* **BvSAT** *is* NP*-complete.* ■

Furthermore, we conclude NP-completeness for the class of **BvSAT** problems where the bitwise bitvector function is chosen to be the constant zero of arity $k \in \mathbb{N}_+$ and of width $n \in \mathbb{N}_+$ on bitvectors of width n. Satisfiability of such problems solely depends on the set I of disequality constraints. For each $k, n \in \mathbb{N}_+$, let $\mathcal{F}^k_{\mathsf{0}_{[n]}}$ denote the k-ary bitvector function defined by $\mathcal{F}^k_{\mathsf{0}_{[n]}}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) := \mathsf{0}_{[n]}$ for all $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$. Then we have:

Proposition 4.21 (NP-Completeness) $\mathsf{BvSAT}(\mathcal{F}^{\bullet}_{0_{[\,\bullet\,]}}, \cdot)$ *is* NP*-complete.*

Proof: Let $k \in \mathbb{N}_+$. Obviously, $\mathsf{BvSAT}(\mathcal{F}^{k}_{0_{[\,\bullet\,]}}, \cdot)$ is in NP. We show that **3-COLORING** is polytime reducible to $\mathsf{BvSAT}(\mathcal{F}^{k}_{0_{[2]}}, \cdot)$. Therefore, let $E \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$. We show that for each undirected graph $G(\{1,\ldots,k\}, E)$ a set I of disequality constraints can be constructed such that $G(\{1,\ldots,k\}, E)$ can be colored with colors $\{0,1,2\}$ such that adjacent nodes have different colors if and only if $\mathsf{BvSAT}(\mathcal{F}^{k}_{0_{[2]}}, I)$ is satisfiable.

Let $I := E \cup \{\, \{i, k+1\} \mid i \in \{1,\ldots,k\} \,\}$. Assume, there exists a 3-coloring of $G(\{1,\ldots,k\}, E)$ with colors $\{0,1,2\}$. For all $i \in \{1,\ldots,k\}$ let

$$\boldsymbol{x}^{i}_{[2]} \;:=\; \mathsf{int2bv}(2, c_i) \in \mathbb{B}_{[2]}\,,$$

where c_i is the color of node i in $G(\{1,\ldots,k\}, E)$. Let $\boldsymbol{x}^{k+1}_{[2]} := \mathtt{11} \in \mathbb{B}_{[2]}$. Then $\boldsymbol{x}^{i}_{[2]} \neq \boldsymbol{x}^{j}_{[2]}$ for all $\{i,j\} \in E$, and $\boldsymbol{x}^{k+1}_{[2]} \neq \boldsymbol{x}^{i}_{[2]}$ for all $i \in \{1,\ldots,k\}$. Then $\langle \boldsymbol{x}^{1}_{[2]}, \ldots, \boldsymbol{x}^{k+1}_{[2]} \rangle$ is a satisfying solution of $\mathsf{BvSAT}(\mathcal{F}^{k}_{0_{[2]}}, I)$.

Now, let $\boldsymbol{x}^{1}_{[2]}, \ldots, \boldsymbol{x}^{k+1}_{[2]} \in \mathbb{B}_{[2]}$ such that $\langle \boldsymbol{x}^{1}_{[2]}, \ldots, \boldsymbol{x}^{k+1}_{[2]} \rangle$ is a satisfying solution of $\mathsf{BvSAT}(\mathcal{F}^{k}_{0_{[2]}}, I)$. Then $\boldsymbol{x}^{k+1}_{[2]} \neq \boldsymbol{x}^{i}_{[2]}$ for all $i \in \{1,\ldots,k\}$. $G(\{1,\ldots,k\}, E)$ can be colored by assigning color $\boldsymbol{x}^{i}_{[2]}$ to node $i \in \{1,\ldots,k\}$, using only the three colors of $\{\mathtt{00}, \mathtt{01}, \mathtt{10}, \mathtt{11}\} \setminus \{\boldsymbol{x}^{k+1}_{[2]}\}$. ■

NP-completeness of $\mathsf{BvSAT}(\mathcal{F}^{k}_{0_{[\,\bullet\,]}}, \cdot)$ can also be concluded from the following corollary, which provides a satisfiability criterion for instances of $\mathsf{BvSAT}(\mathcal{F}^{k}_{0_{[n]}}, I)$ by relating the width n of the bitvector domains to the chromatic number of a finite graph constructed from I.

For each undirected graph $G(V, E)$ with vertices V and edges E, let $\chi(G)$ denote the chromatic number of G, i.e. the minimum number of colors which is needed in order to color the vertices of G such that no two adjacent vertices have the same color. Then we have:

Corollary 4.22 (Satisfiability of Sets of Disequality Constraints) *Let* $k \in \mathbb{N}_+$ *and let* $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$*. Let* $n \in \mathbb{N}_+$*. Then the following holds:*

$$\mathsf{BvSAT}(\mathcal{F}^{k}_{0_{[n]}}, I) \text{ is satisfiable} \quad\Longleftrightarrow\quad 2^n \;\geq\; \chi\big(\, G(\{1,\ldots,k\}, I) \,\big)$$

Proof: Satisfiability of $\mathsf{BvSAT}(\mathcal{F}^{k}_{0_{[n]}}, I)$ solely depends on satisfiability of I. A satisfying solution of I yields a chromatic coloring of $G(\{1,\ldots,k\}, I)$, and a chromatic coloring of G can be transformed into a satisfying solution of I by encoding the color values as bitvectors. Here, a minimum of at least $\log\, \chi(G)$ bits is needed to be able to encode $\chi(G)$ different colors. ■

Proposition 4.21 can also be concluded from Corollary 4.22 because the general problem of deciding whether the chromatic number of an arbitrary graph is less than a given $n \in \mathbb{N}_+$ is NP-complete for $n > 2$.

4.6 The Enhanced BvSAT Problem and Bitwise Bitvector Functions

Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$ be a set of disequality constraints for k bitvectors. Let $n := \lfloor \log\ \chi(G(\{1,\ldots,k\}, I)) \rfloor$. Corollary 4.22 yields that **BvSAT** problems for $(\mathcal{F}^k_{0_{[i]}})_{i\in\mathbb{N}_+}$ and I can be characterized as follows:

$$\forall\ i \geq n\ :\ \mathbf{BvSAT}(\mathcal{F}^k_{0_{[i]}}, I)\ \ \textit{is satisfiable}$$

and

$$\forall\ i < n\ :\ \mathbf{BvSAT}(\mathcal{F}^k_{0_{[i]}}, I)\ \ \textit{is unsatisfiable}$$

The enhanced satisfiability problem for bitvector functions is monotone in the width i of the bitvectors domains if **BvSAT** is restricted to a family $(\mathcal{F}^k_{0_{[i]}})_{i\in\mathbb{N}_+}$ of corresponding zero functions and to a fixed set I of disequality constraints.

Example 4.23 (Monotony of BvSAT) *Let $k := 3$ and let $I := \{\, \{1,2\}, \{2,3\}, \{3,1\} \,\}$. Then $\chi(G(\{1,2,3\}, I)) = 3$ and $\log\ \chi(G(\{1,2,3\}, I)) = 2$. Hence, $\mathbf{BvSAT}(\mathcal{F}^3_{0_{[1]}}, I)$ is unsatisfiable because, for each satisfying solution $\langle \boldsymbol{x}^1_{[i]}, \boldsymbol{x}^2_{[i]}, \boldsymbol{x}^3_{[i]} \rangle$ of I, the bitvectors $\boldsymbol{x}^1_{[i]}, \boldsymbol{x}^2_{[i]}, \boldsymbol{x}^3_{[i]} \in \mathbb{B}_{[i]}$ must be mutually different, which is not possible for $i = 1$, but for all $i \geq 2$, i.e. $\mathbf{BvSAT}(\mathcal{F}^3_{0_{[i]}}, I)$ is satisfiable for all $i \geq 2$.* □

In the remaining sections, satisfiability of classes $\mathbf{BvSAT}(\mathcal{F}_{[\,\cdot\,]}, \cdot)$ of **BvSAT** problems is examined where $(\mathcal{F}_{[i]})_{i\in\mathbb{N}_+}$ is an arbitrary family of corresponding k-ary bitwise bitvector functions for a fixed arity $k \in \mathbb{N}_+$. We present satisfiability criteria which reveal how satisfiability depends on the widths of the bitvector domains. As a first step, the monotony characterization given above is generalized for arbitrary bitwise bitvector functions.

Proposition 4.24 (Monotony of BvSAT) *Let $k \in \mathbb{N}_+$ and $(\mathcal{F}_{[i]})_{i\in\mathbb{N}_+}$ be a family of corresponding k-ary bitwise bitvector functions. Let $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$. Then $\mathbf{BvSAT}(\mathcal{F}_{[\,\cdot\,]}, I)$ is monotone in the widths of the bitvector functions, i.e. for each $n \in \mathbb{N}_+$ the following hold:*

$$\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)\ \textit{is satisfiable} \quad\Longrightarrow\quad \forall\ i \geq n\ :\ \mathbf{BvSAT}(\mathcal{F}_{[i]}, I)\ \textit{is satisfiable}$$

and

$$\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)\ \textit{is unsatisfiable} \quad\Longrightarrow\quad \forall\ i \leq n\ :\ \mathbf{BvSAT}(\mathcal{F}_{[i]}, I)\ \textit{is unsatisfiable}$$

Proof: Assume, $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$ is satisfiable, and let $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ be a satisfying solution. Let $i \in \mathbb{N}_+$ with $i \geq n$. According to Theorem 4.9 then $\langle \mathsf{signExt}_{[i]}(\boldsymbol{x}^1_{[n]}), \ldots, \mathsf{signExt}_{[i]}(\boldsymbol{x}^k_{[n]}) \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[i]})$ and, according to Corollary 4.18, furthermore satisfies I. Hence, $\langle \mathsf{signExt}_{[i]}(\boldsymbol{x}^1_{[n]}), \ldots, \mathsf{signExt}_{[i]}(\boldsymbol{x}^k_{[n]}) \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[i]}, I)$.

Now let $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$ be unsatisfiable, and let $i \in \mathbb{N}_+$ with $i \leq n$. Assume that $\mathbf{BvSAT}(\mathcal{F}_{[i]}, I)$ is satisfiable. Then, according to the first part of the proof, $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$ is satisfiable. ↯ ■

The proof of Proposition 4.24 is based on the following characterization of signed extensions of satisfying solutions of **BvSAT** problems.

Corollary 4.25 (Bitwise BvSAT Problems and Signed Extension) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$. Let $(\mathcal{F}_{[i]})_{i\in\mathbb{N}_+}$ be a family of corresponding k-ary bitwise bitvector functions, and let $m, n \in \mathbb{N}_+$ with $n \geq m$. Then for all $\boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \in \mathbb{B}_{[m]}$ the following holds:*

$$\langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle \in \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[m]}, I) \iff \langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[m]}), \ldots, \mathsf{signExt}_{[n]}(\boldsymbol{x}^k_{[m]}) \rangle \in \mathcal{S}_{\mathsf{BvSAT}}(\mathcal{F}_{[n]}, I)$$

Proof: Follows immediately from Corollary 4.10. ■

If a bitwise $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$ problem is satisfiable, then the monotony characterization given in Proposition 4.24 implies that there exists a uniquely determined width $m \in \mathbb{N}_+$ such that all corresponding bitwise satisfiability problems of width greater or equal m are satisfiable and all corresponding bitwise satisfiability problems of width less than m are unsatisfiable.

Theorem 4.26 (Satisfiability of BvSAT) *Let $k \in \mathbb{N}_+$ and $(\mathcal{F}_{[i]})_{i\in\mathbb{N}_+}$ be a family of corresponding k-ary bitwise bitvector functions. Let $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$. Then either $\mathbf{BvSAT}(\mathcal{F}_{[i]}, I)$ is unsatisfiable for all $i \in \mathbb{N}_+$, or there exists $m \in \mathbb{N}_+$ such that*

$$\forall\ i \geq m\ :\ \mathbf{BvSAT}(\mathcal{F}_{[i]}, I) \text{ is satisfiable}$$

and

$$\forall\ i < m\ :\ \mathbf{BvSAT}(\mathcal{F}_{[i]}, I) \text{ is unsatisfiable}$$

Proof: Follows immediately from Proposition 4.24. ■

For each bitwise bitvector function $\mathcal{F}_{[n]}$ this characteristic width m is called the *satisfiability threshold* of $\mathcal{F}_{[n]}$ and I. It is denoted by $\mu(\mathcal{F}_{[n]}, I)$ and defined as follows.

Definition 4.27 (Satisfiability Threshold) *Let $k \in \mathbb{N}_+$ and $(\mathcal{F}_{[i]})_{i\in\mathbb{N}_+}$ be a family of corresponding k-ary bitwise bitvector functions. Let $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$ and let $n \in \mathbb{N}_+$. The **satisfiability threshold** $\mu(\mathcal{F}_{[n]}, I)$ for $\mathcal{F}_{[n]}$ and I is defined as*

$$\mu(\mathcal{F}_{[n]}, I) := \min\{\, i \in \mathbb{N}_+ \mid \mathbf{BvSAT}(\mathcal{F}_{[i]}, I) \text{ is satisfiable} \,\}$$

if there exists $i \in \mathbb{N}_+$ such that $\mathbf{BvSAT}(\mathcal{F}_{[i]}, I)$ is satisfiable, and

$$\mu(\mathcal{F}_{[n]}, I) := \infty$$

if $\mathbf{BvSAT}(\mathcal{F}_{[i]}, I)$ is unsatisfiable for all $i \in \mathbb{N}_+$. ■

Consequently, the computational complexity of determining the satisfiability threshold can be characterized as follows ($\mathcal{F}_{[n]}$ denoting arbitrary bitwise bitvector functions and I denoting arbitrary sets of disequality constraints):

Proposition 4.28 (NP-Completeness) *Deciding if on input $\mathcal{F}_{[n]}$, I and $m \in \mathbb{N}_+$ the satisfiability threshold $\mu(\mathcal{F}_{[n]}, I) = m$ is* NP*-complete.* ■

Proposition 4.29 (NP-Completeness) *Deciding if on input $\mathcal{F}_{[n]}$, I and $m \in \mathbb{N}_+$ the satisfiability threshold $\mu(\mathcal{F}_{[n]}, I) \leq m$ is* NP*-complete.* ■

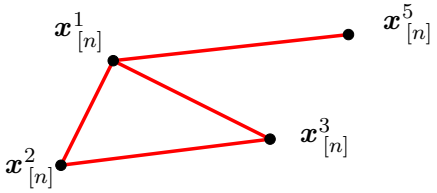
4.7 Disequality Graphs

As we have seen in Corollary 4.22 and Example 4.23, a set I of disequality constraints for k bitvectors can be represented by a finite undirected graph which is called the *disequality graph* corresponding to k and I.

Definition 4.30 (Disequality Graphs) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ be a set of disequality constraints for k bitvectors. The* ***disequality graph*** *$\mathcal{G}_{\neq}(k, I)$ for k and I is defined as the undirected graph $G(\{1, \ldots, k\}, I)$ with vertices $1, \ldots, k$ and edges I.* ■

Disequality constraints of I correspond to edges of $\mathcal{G}_{\neq}(k, I)$, and the vertices of $\mathcal{G}_{\neq}(k, I)$ represent an ordered set of bitvector variables. When we depict disequality graphs, vertices will be labeled with bitvector variables $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}$ rather than with natural numbers $1, \ldots, k$ whenever we want to point out that in the current context disequality constraints for bitvectors of a specific width n are considered.

Example 4.31 (Disequality Graph) *Let $I := \{\, \{1,2\}, \{2,3\}, \{1,3\}, \{1,5\} \,\}$ be a set of disequality constraints for 5 bitvectors. Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^5_{[n]} \in \mathbb{B}_{[n]}$. The corresponding disequality graph $\mathcal{G}_{\neq}(5, I)$ is shown below:*



In this case, $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^5_{[n]} \rangle$ satisfies I if and only if $\boldsymbol{x}^1_{[n]} \neq \boldsymbol{x}^2_{[n]}$, $\boldsymbol{x}^2_{[n]} \neq \boldsymbol{x}^3_{[n]}$, $\boldsymbol{x}^1_{[n]} \neq \boldsymbol{x}^3_{[n]}$ and $\boldsymbol{x}^1_{[n]} \neq \boldsymbol{x}^5_{[n]}$. □

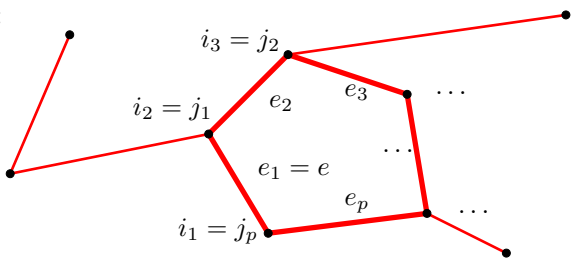
The next theorem relates satisfiability of disequality constraints to structural properties of the corresponding disequality graph.

If there exists a cycle in the disequality graph, then satisfying solutions of the disequality constraints have the following property:

Theorem 4.32 (Disequality Constraints and Cycles in Disequality Graphs) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$ be a set of disequality constraints for k bitvectors. Let $\mathcal{G}_{\neq}(k,I)$ be the corresponding disequality graph, and let $e \in I$. Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $\langle \boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \rangle$ satisfies e. Assume that e lies on a cycle γ in $\mathcal{G}_{\neq}(k,I)$. Then there exists $e' \in I$ on the same cycle γ, with $e' \neq e$, and $\langle \boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \rangle$ also satisfies e'.*

Proof: Let $k \in \mathbb{N}_+$ and $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$ such that there exists a cycle γ in $\mathcal{G}_{\neq}(k,I)$. Let $e \in I$ such that e lies on γ. Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $\langle \boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \rangle$ satisfies e. Without loss of generality we can assume γ to be a simple cycle, consisting of p edges $e_1,\ldots,e_p$, with $p \in \mathbb{N}_+$. Then $p \geq 3$ and there exist $i_1,\ldots,i_p \in \{1,\ldots,k\}$ and $j_1,\ldots,j_p \in \{1,\ldots,k\}$ such that $e_1 = \{i_1,j_1\},\ldots,e_p = \{i_p,j_p\}$ and $j_q = i_{q+1}$ for all $q \in \mathbb{N}_+$ with $1 \leq q < p$, and $j_p = i_1$ and $e = e_1$, as illustrated below.

$\mathcal{G}_{\neq}(k,I)$:



According to Lemma 4.17, there exists a column $l \in \{0,\ldots,n-1\}$, such that $\langle \boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \rangle[l]$ satisfies e. Then $\boldsymbol{x}^{i_1}_{[n]}[l] \neq \boldsymbol{x}^{j_1}_{[n]}[l]$, i.e.

$$\boldsymbol{x}^{i_1}_{[n]}[l] = \mathsf{neg}(\boldsymbol{x}^{j_1}_{[n]}[l]) = \mathsf{neg}(\boldsymbol{x}^{i_2}_{[n]}[l]).$$

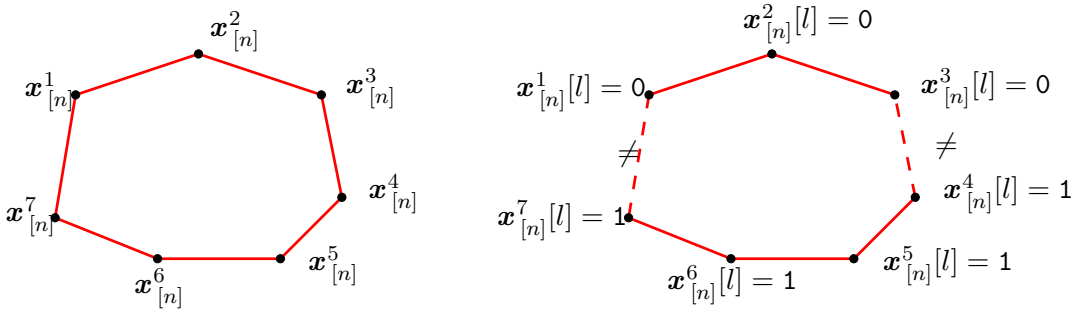
Assume, $\langle \boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \rangle[l]$ does not satisfy e_q for all $1 < q \leq p$. Then $\boldsymbol{x}^{i_q}_{[n]}[l] = \boldsymbol{x}^{j_q}_{[n]}[l]$ for all $1 < q \leq p$, i.e. $\boldsymbol{x}^{i_q}_{[n]}[l] = \boldsymbol{x}^{i_{q+1}}_{[n]}[l]$ for all $1 < q < p$. But then we have:

$$\begin{aligned}
\boldsymbol{x}^{i_1}_{[n]}[l] &= \mathsf{neg}(\boldsymbol{x}^{i_2}_{[n]}[l]) \\
&= \mathsf{neg}(\boldsymbol{x}^{i_3}_{[n]}[l]) \\
&= \ldots \\
&= \mathsf{neg}(\boldsymbol{x}^{i_p}_{[n]}[l]) \\
&= \mathsf{neg}(\boldsymbol{x}^{j_p}_{[n]}[l]) \\
&= \mathsf{neg}(\boldsymbol{x}^{i_1}_{[n]}[l]) \quad \lightning
\end{aligned}$$

Therefore, there must exist $q \in \mathbb{N}_+$ with $1 < q \leq p$ such that $\boldsymbol{x}^{i_q}_{[n]}[l] \neq \boldsymbol{x}^{j_q}_{[n]}[l]$. But then $\langle \boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \rangle[l]$ satisfies e_q, and thus $\langle \boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \rangle$ satisfies e_q. ■

The reasoning which is used in the proof of Theorem 4.32 is based on the following idea: if all vertices of a cycle are colored using only two colors 0 and 1, and if there exists at least one edge on this cycle which carries different colors at both ends, then it is impossible to color all the remaining edges in a way such that both ends always carry the same color. This idea is further illustrated by the following example.

Example 4.33 (Disequality Constraints and Cycles in Disequality Graphs) *Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^7_{[n]} \in \mathbb{B}_{[n]}$. Let $I = \{\{1,2\}, \{2,3\}, \{3,4\}, \{4,5\}, \{5,6\}, \{6,7\}, \{7,1\}\}$. The corresponding disequality graph $\mathcal{G}_{\neq}(7, I)$ is shown below on the left.*



Let $l \in \{0, \ldots, n-1\}$ and assume $\boldsymbol{x}^1_{[n]}[l] = 0$, $\boldsymbol{x}^2_{[n]}[l] = 0$, $\boldsymbol{x}^3_{[n]}[l] = 0$, $\boldsymbol{x}^4_{[n]}[l] = 1$, $\boldsymbol{x}^5_{[n]}[l] = 1$, $\boldsymbol{x}^6_{[n]}[l] = 1$, $\boldsymbol{x}^7_{[n]}[l] = 1$. Then $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^7_{[n]} \rangle$ satisfies the disequality constraint $\{1,7\} \in I$, and $\{1,7\}$ lies on a cycle in $\mathcal{G}_{\neq}(7, I)$. Shown on the right in the picture above, each node i is labeled with the value of the corresponding bitvector $\boldsymbol{x}^i_{[n]}$ at index l. As can be seen, disequality $\{3,4\} \in I$ is also satisfied by $\langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^7_{[n]} \rangle$. □

Theorem 4.32 is used to further characterize existence of satisfying solutions of sets of disequality constraints. The width i of the bitvector domains $\mathbb{B}_{[i]}$ for which a satisfying solution of a given set I exists is related to connectivity properties of a corresponding connected disequality graph in the following way (arbitrary disequality graphs are considered in Theorem 4.35):

Theorem 4.34 (Disequality Constraints and Connected Disequality Graphs) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{e \subseteq \{1, \ldots, k\} \mid |e| = 2\}$ with $I \neq \emptyset$, such that the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$ is connected. Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ and $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$. Then the following holds:*

$$S \text{ satisfies } I \implies \exists\, i_1, \ldots, i_{k-1} \in \{0, \ldots, n-1\} \,:\, S[i_1] \otimes \ldots \otimes S[i_{k-1}] \text{ satisfies } I$$

Proof: Let $k \in \mathbb{N}_+$ and let $I \subseteq \{e \subseteq \{1, \ldots, k\} \mid |e| = 2\}$ such that $I \neq \emptyset$ and such that the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$ is connected. Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ satisfies I.

Let $p := |I|$ and let $e_1, \ldots, e_p \subseteq \{1, \ldots, k\}$ with $|e_i| = 2$ for all $i \in \{1, \ldots, p\}$, such that $I = \{e_1, \ldots, e_p\}$. According to Lemma 4.17, there exist columns $c_1, \ldots, c_p \in \{0, \ldots, n-1\}$ such that for each $i \in \{1, \ldots, p\}$ disequality constraint e_i is satisfied by $S[c_i]$.

We construct a selection of $k-1$ columns of $c_1, \ldots, c_p$ by determining $i_1, \ldots, i_{k-1} \in \{1, \ldots, p\}$ such that $S[c_{i_1}] \otimes \ldots \otimes S[c_{i_{k-1}}]$ satisfies I. Therefore, initially let $J := I$. Then $i_1, \ldots, i_{k-1}$ are computed in the following way:

1. The disequality graph $\mathcal{G}_{\neq}(k, J)$ is connected, and therefore $|J| \geq k-1$. Let $m := |J|$. Because $J \subseteq I$, there exist $i_1, \ldots, i_m \in \{1, \ldots, p\}$ such that $J = \{e_{i_1}, \ldots, e_{i_m}\}$. If $|J| = k-1$ then stop.

2. Otherwise $|J| > k-1$. Then there exists a cycle γ in $\mathcal{G}_{\neq}(k, J)$. Without loss of generality, γ is a simple cycle. Let $l \in \{1, \ldots, p\}$ such that $e_l \in J$ and e_l is an edge on γ. Then $S[c_l]$ satisfies e_l.

3. (a) Because e_l is on cycle γ, according to Theorem 4.32 there exists $r \in \{1, \ldots, p\}$ such that $e_r \in J$ and e_r is on γ with $e_l \neq e_r$, and e_r is also satisfied by $S[c_l]$.

   (b) Remove e_r from J, i.e. let $J := J \setminus \{e_r\}$.

   (c) Because e_r was on a cycle in $\mathcal{G}_{\neq}(k, J \cup \{e_r\})$, we can conclude that $\mathcal{G}_{\neq}(k, J)$ is still connected. If edge e_l is still on a cycle in $\mathcal{G}_{\neq}(k, J)$, then let γ be such cycle and repeat steps 3(a) to 3(c).

4. Continue with step 1.

Iteration of steps 3(a) to 3(c) and of steps 1 to 4 terminates because $|J|$ is decreased in each loop. According to the termination criterion in step 1, we then have $|J| = k-1$ and thus $m = k-1$. Let $S' := S[c_{i_1}] \otimes \ldots \otimes S[c_{i_m}]$ for those $i_1, \ldots, i_m$ selected in the last execution of step 1 before termination. Then S' satisfies J.

Let $r \in \{1, \ldots, p\}$ such that $e_r \in I \setminus J$. Then e_r was removed from the initial J at some point by execution of step 3(b), and according to prior execution of steps 2 and 3(a) there exists $l \in \{1, \ldots, p\}$ such that $e_l \in I$ with $e_l \neq e_r$, and $S[c_l]$ satisfies e_r. For this e_l, we have $e_l \in J$, as each edge e_l which is selected in step 2 is *never* removed from J because, after execution of step 4, e_l does not reside on a cycle anymore, and in step 3(b) only edges on cycles are removed. Thus there exists $j \in \{1, \ldots, m\}$ such that $i_j = l$, and therefore $S[c_{i_1}] \otimes S[c_{i_2}] \otimes \ldots \otimes S[c_{i_m}]$ satisfies e_r.

Hence, S' satisfies I. ■

Theorem 4.34 states that, if a set I of disequality constraints for k bitvectors is satisfiable for bitvectors of a certain width n, and if the corresponding disequality graph is connected, then for each satisfying solution S of I of width n there exists a satisfying solution of I which

- consists of bitvectors of width $k-1$ and
- which is composed exclusively of columns of S.

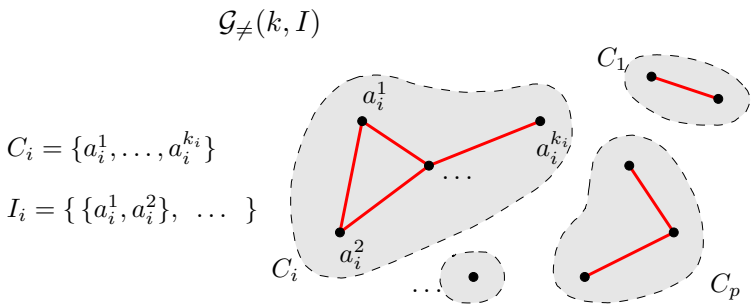
As a next step, we give an analogous characterization for the case of arbitrary disequality graphs, i.e. connectivity of $\mathcal{G}_{\neq}(k, I)$ is not required anymore.

Theorem 4.35 (Disequality Constraints and Disequality Graphs) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\} \neq \emptyset$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$. Let $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$. Then the following holds:*

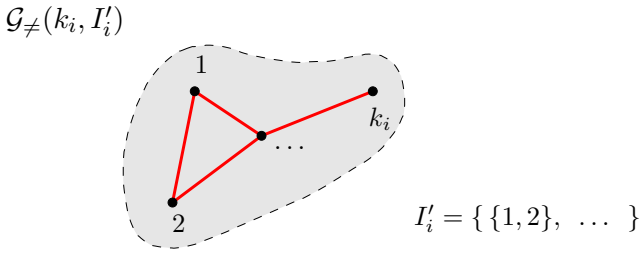
$$S \text{ satisfies } I \implies \exists\, i_1, \ldots, i_{k-p} \in \{0, \ldots, n-1\} \;:\; S[i_1] \otimes \ldots \otimes S[i_{k-p}] \text{ satisfies } I$$

Proof: Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1,\ldots,k\} \mid |e| = 2 \,\}$ such that $I \neq \emptyset$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ satisfies I.

Let $C_1, \ldots, C_p \subseteq \{1, \ldots, k\}$ be the connected components of $\mathcal{G}_{\neq}(k, I)$. For each $i \in \{1, \ldots, p\}$ let $k_i := |C_i|$ and let $a^1_i, \ldots, a^{k_i}_i \in \{1, \ldots, k\}$ such that $C_i = \{a^1_i, \ldots, a^{k_i}_i\}$. Furthermore, let $I_i := \{\, e \in I \mid e \subseteq C_i \,\}$ be the set of disequality constraints related to component C_i.



Because S satisfies I, and as $I_i \subseteq I$, we also have that S satisfies I_i. Let $S_i := \langle \boldsymbol{x}^{a^1_i}_{[n]}, \ldots, \boldsymbol{x}^{a^{k_i}_i}_{[n]} \rangle$ and let $I'_i := \{\, \{\alpha, \beta\} \mid \alpha, \beta \in \{1, \ldots, k_i\} \wedge \{a^\alpha_i, a^\beta_i\} \in I_i \,\}$. Then S_i satisfies I'_i and $\mathcal{G}_{\neq}(k_i, I'_i)$ is connected.



Then according to Theorem 4.34 there exist columns $j^1_i, \ldots, j^{k_i-1}_i \in \{0, \ldots, n-1\}$ such that $S_i[j^1_i] \otimes \ldots \otimes S_i[j^{k_i-1}_i]$ satisfies I'_i. Then $S[j^1_i] \otimes \ldots \otimes S[j^{k_i-1}_i]$ satisfies I_i, and thus in summary

$$S[j^1_1] \otimes \ldots \otimes S[j^{k_1-1}_1] \;\otimes\; S[j^1_2] \otimes \ldots \otimes S[j^{k_2-1}_2] \;\otimes\; \ldots \;\otimes\; S[j^1_p] \otimes \ldots \otimes S[j^{k_p-1}_i]$$

satisfies all sets $I_1, \ldots, I_p$ of disequality constraints, and hence satisfies $\bigcup_{i \in \{1,\ldots,p\}} I_i = I$.

Furthermore we have:

$$\begin{aligned}\sum_{i\in\{1,\ldots,p\}}(k_i-1) &= \sum_{i\,\in\,\{1,\ldots,p\}} k_i \;-\; \sum_{i\,\in\,\{1,\ldots,p\}} 1\\ &= \sum_{i\,\in\,\{1,\ldots,p\}} k_i \;-\; p\\ &= k \;-\; p\end{aligned}$$

and hence Theorem 4.35 holds. ■

4.8 Reduction Theorems and Proofs

The main objective of this section is to characterize the correlation between corresponding bitwise **BvSAT** instances of different widths. We analyze in which way satisfiability of a given **BvSAT**$(\mathcal{F}_{[n]}, I)$ problem depends on the width n of the bitvector domain $\mathbb{B}_{[n]}$.

As pointed out in the previous sections, **BvSAT** is NP-complete. Any decision procedure known so far which determines satisfiability of arbitrary instances **BvSAT**$(\mathcal{F}_{[n]}, I)$ has (and will have, unless P = NP) a complexity which is at least super-polynomial in n. However, a large class of hardware verification problems can be characterized in terms of satisfiability problems for bitvector functions and disequality constraints, and decision procedures for **BvSAT** problems are widely (and successfully) applied in various fields of industrial hardware verification.

We present an abstraction criterion which relates satisfiability of bitwise **BvSAT** problems to the width of the respective bitvector domains. This criterion allows to determine satisfiability of a specific bitwise instance **BvSAT**$(\mathcal{F}_{[n]}, I)$ for given $\mathcal{F}_{[n]}$, I and $n \in \mathbb{N}_+$ by deciding satisfiability of a corresponding "smaller" bitwise instance **BvSAT**$(\mathcal{F}_{[m]}, I)$ with $m \leq n$ and $\mathcal{F}_{[m]} \simeq \mathcal{F}_{[n]}$. We furthermore present an analogous criterion for satisfying solutions which shows how to relate a satisfying solution of such a smaller instance to a satisfying solution of the original problem.

Example 4.36 (Satisfiability of Corresponding Bitwise BvSAT Problems) *Let $n \in \mathbb{N}_+$ with $n \geq 2$ and let $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \to \mathbb{B}_{[n]}$ be an arbitrary ternary bitwise bitvector function on bitvectors of width n. Let $I := \{\,\{1,2\},\{2,3\}\,\}$, and let $\boldsymbol{x}^1_{[n]}, \boldsymbol{x}^2_{[n]}, \boldsymbol{x}^3_{[n]} \in \mathbb{B}_{[n]}$ such that $\langle \boldsymbol{x}^1_{[n]}, \boldsymbol{x}^2_{[n]}, \boldsymbol{x}^3_{[n]} \rangle$ is a satisfying solution of* **BvSAT**$(\mathcal{F}_{[n]}, I)$*, i.e.*

$$\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \boldsymbol{x}^2_{[n]}, \boldsymbol{x}^3_{[n]}) = \mathsf{0}_{[n]} \quad \textit{and} \quad \boldsymbol{x}^1_{[n]} \neq \boldsymbol{x}^2_{[n]} \quad \textit{and} \quad \boldsymbol{x}^2_{[n]} \neq \boldsymbol{x}^3_{[n]}.$$

Then according to Proposition 4.15 there exist two columns $i, j \in \{0, \ldots, n-1\}$ such that $\boldsymbol{x}^1_{[n]}[i] \neq \boldsymbol{x}^2_{[n]}[i]$ and $\boldsymbol{x}^2_{[n]}[j] \neq \boldsymbol{x}^3_{[n]}[j]$. Let $\boldsymbol{y}^1_{[2]}, \boldsymbol{y}^2_{[2]}, \boldsymbol{y}^3_{[2]} \in \mathbb{B}_{[2]}$ such that

$$\boldsymbol{y}^1_{[2]} := \boldsymbol{x}^1_{[n]}[i] \otimes \boldsymbol{x}^1_{[n]}[j], \qquad \boldsymbol{y}^2_{[2]} := \boldsymbol{x}^2_{[n]}[i] \otimes \boldsymbol{x}^2_{[n]}[j], \qquad \boldsymbol{y}^3_{[2]} := \boldsymbol{x}^3_{[n]}[i] \otimes \boldsymbol{x}^3_{[n]}[j].$$

Let $\mathcal{F}_{[2]}$ denote the corresponding bitwise bitvector function of width 2 with $\mathcal{F}_{[2]} \simeq \mathcal{F}_{[n]}$. Then $\mathcal{F}_{[2]}(\boldsymbol{y}^1_{[2]}, \boldsymbol{y}^2_{[2]}, \boldsymbol{y}^3_{[2]}) = \mathsf{0}_{[2]}$ and $\boldsymbol{y}^1_{[2]} \neq \boldsymbol{y}^2_{[2]}$ and $\boldsymbol{y}^2_{[2]} \neq \boldsymbol{y}^3_{[2]}$, i.e. $\langle \boldsymbol{y}^1_{[2]}, \boldsymbol{y}^2_{[2]}, \boldsymbol{y}^3_{[2]} \rangle$ is a satisfying solution of **BvSAT**$(\mathcal{F}_{[2]}, I)$.

Now let $\boldsymbol{x}^1_{[2]}, \boldsymbol{x}^2_{[2]}, \boldsymbol{x}^3_{[2]} \in \mathbb{B}_{[2]}$ *such that* $\langle \boldsymbol{x}^1_{[2]}, \boldsymbol{x}^2_{[2]}, \boldsymbol{x}^3_{[2]} \rangle$ *is a satisfying solution of* $\mathbf{BvSAT}(\mathcal{F}_{[2]}, I)$*. Then, according to Corollary 4.25,* $\langle \mathsf{signExt}_{[n]}(\boldsymbol{x}^1_{[2]}), \mathsf{signExt}_{[n]}(\boldsymbol{x}^2_{[2]}), \mathsf{signExt}_{[n]}(\boldsymbol{x}^3_{[2]}) \rangle$ *is a satisfying solution of* $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$*. As a result, we have shown that*

$$\mathbf{BvSAT}(\mathcal{F}_{[n]}, \{\{1,2\},\{2,3\}\}) \text{ is satisfiable} \iff \mathbf{BvSAT}(\mathcal{F}_{[2]}, \{\{1,2\},\{2,3\}\}) \text{ is satisfiable}$$

for all $n \geq 2$ *and all bitwise bitvector functions* $\mathcal{F}_{[n]}$*. Accordingly, deciding satisfiability of* $\mathbf{BvSAT}(\mathcal{F}_{[n]}, \{\{1,2\},\{2,3\}\})$ *can always be reduced in a one-to-one fashion to determining satisfiability of* $\mathbf{BvSAT}(\mathcal{F}_{[2]}, \{\{1,2\},\{2,3\}\})$*.* □

Example 4.37 (Satisfiability of Corresponding Bitwise BvSAT Problems) *For* $n \geq 2$ *and arbitrary bitwise bitvector functions* $\mathcal{F}_{[n]}$*, deciding if* $\mathbf{BvSAT}(\mathcal{F}_{[n]}, \{\{1,2\},\{2,3\}\})$ *is satisfiable cannot generally be reduced to determining satisfiability of* $\mathbf{BvSAT}(\mathcal{F}_{[1]}, \{\{1,2\},\{2,3\}\})$ *with* $\mathcal{F}_{[1]} \simeq \mathcal{F}_{[n]}$*. To see this, let* $\mathcal{F}_{[n]} : \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \times \mathbb{B}_{[n]} \longrightarrow \mathbb{B}_{[n]}$ *be defined by*

$$\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \boldsymbol{x}^2_{[n]}, \boldsymbol{x}^3_{[n]}) \;:=\; \mathsf{neg}\big((\mathsf{neg}(\boldsymbol{x}^1_{[n]}) \,\mathsf{and}\, \boldsymbol{x}^2_{[n]} \,\mathsf{and}\, \boldsymbol{x}^3_{[n]}) \;\mathsf{or}\; (\boldsymbol{x}^1_{[n]} \,\mathsf{and}\, \boldsymbol{x}^2_{[n]} \,\mathsf{and}\, \mathsf{neg}(\boldsymbol{x}^3_{[n]}))\big)$$

for all $\boldsymbol{x}^1_{[n]}, \boldsymbol{x}^2_{[n]}, \boldsymbol{x}^3_{[n]} \in \mathbb{B}_{[n]}$*. Then* $\mathcal{F}_{[n]}$ *is bitwise, and* $\mathbf{BvSAT}(\mathcal{F}_{[n]}, \{\{1,2\},\{2,3\}\})$ *is satisfiable, for example by*

$$\boldsymbol{x}^1_{[n]} := \mathtt{0\ldots01}, \quad \boldsymbol{x}^2_{[n]} := \mathtt{1\ldots11} \quad \textit{and} \quad \boldsymbol{x}^3_{[n]} := \mathtt{1\ldots10},$$

while the only satisfying solutions of $\mathcal{F}_{[1]}(\boldsymbol{x}^1_{[1]}, \boldsymbol{x}^2_{[1]}, \boldsymbol{x}^3_{[1]}) = \mathtt{0}_{[1]}$*, with* $\mathcal{F}_{[1]} \simeq \mathcal{F}_{[n]}$*, are*

$$\boldsymbol{x}^1_{[1]} := \mathtt{0},\ \boldsymbol{x}^2_{[1]} := \mathtt{1},\ \boldsymbol{x}^3_{[1]} := \mathtt{1}, \quad \textit{and} \quad \boldsymbol{x}^1_{[1]} := \mathtt{1},\ \boldsymbol{x}^2_{[1]} := \mathtt{1},\ \boldsymbol{x}^3_{[1]} := \mathtt{0},$$

neither of which satisfies all disequalities induced by I*, i.e.* $\mathbf{BvSAT}(\mathcal{F}_{[1]}, I)$ *is unsatisfiable.* □

Examples 4.36 and 4.37 show that the satisfiability threshold $\mu(\mathcal{F}_{[n]}, I)$ for the specific $\mathcal{F}_{[n]}$ and I which are defined in Example 4.37 is 2. In Propositions 4.28 and 4.29 we have seen how general computation of $\mu(\mathcal{F}_{[n]}, I)$ is related to the NP-complete **BvSAT** problem.

Yet, Example 4.36 also yields the more general statement that for the mentioned I and for any bitwise bitvector function $\mathcal{F}_{[n]}$ of arity 3 and arbitrary width $n \in \mathbb{N}_+$ the $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$ problem is satisfiable if and only of the corresponding **BvSAT** problem of width 2 is satisfiable. That is, there possibly exists a bitwise bitvector function $\mathcal{F}_{[n]}$ for which

$$\mathbf{BvSAT}(\mathcal{F}_{[n]}, \{\{1,2\},\{2,3\}\}) \text{ is satisfiable} \iff \mathbf{BvSAT}(\mathcal{F}_{[1]}, \{\{1,2\},\{2,3\}\}) \text{ is satisfiable},$$

holds, but as we have seen, this is not the case for all bitwise $\mathcal{F}_{[n]}$. However, the following holds:

$$\forall \;\; \textit{bitwise } \mathcal{F}_{[n]} \textit{ of arity } 3 \;:\; \mu(\mathcal{F}_{[n]}, \{\{1,2\},\{2,3\}\}) \leq 2\,,$$

and furthermore we know that 2 is the minimum bitvector width for which this is true. This concept is captured by the following definition.

Definition 4.38 (Global Satisfiability Threshold) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$. The **global satisfiability threshold** $\nu(k, I)$ for k-ary bitwise bitvector functions and I is defined as:*

$$\nu(k, I) := \max \{\, \mu(\mathcal{F}_{[n]}, I) \mid n \in \mathbb{N}_+ \wedge \mathcal{F}_{[n]}\ k\text{-}ary \text{ and } bitwise \wedge \mathbf{BvSAT}(\mathcal{F}_{[n]}, I)\ satisfiable \,\}$$

■

The main contribution of this chapter is the result that the global satisfiability threshold $\nu(k, I)$ is always a well-defined natural number, and that, although general computation of $\mu(\mathcal{F}_{[n]}, I)$ is related to NP-completeness as stated before, $\nu(k, I)$ can be efficiently computed for each $k \in \mathbb{N}_+$ and for arbitrary sets I of disequality constraints. This is shown in the rest of this section. First, we address the case that for a given I the disequality graph $\mathcal{G}_{\neq}(k, I)$ is connected.

Theorem 4.39 (Width Reduction) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ such that the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$ is connected. Let $n \in \mathbb{N}_+$ and let*

$$m := \min(n, \max(1, k-1))$$

Then for each family $(\mathcal{F}_{[i]})_{i \in \mathbb{N}_+}$ of corresponding k-ary bitwise bitvector functions the following holds:

$$\mathbf{BvSAT}(\mathcal{F}_{[n]}, I) \ \textit{satisfiable} \quad \Longleftrightarrow \quad \mathbf{BvSAT}(\mathcal{F}_{[m]}, I) \ \textit{satisfiable}$$

Proof: Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ be a set of disequality constraints for k bitvectors such that the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$ is connected. Let $n \in \mathbb{N}_+$ and let $\mathcal{F}_{[n]}$ be an arbitrary k-ary bitwise bitvector function on bitvectors of width n. Let $m := \min(n, \max(1, k-1))$, and let $\mathcal{F}_{[m]}$ be the corresponding k-ary bitwise bitvector function on bitvectors of width m with $\mathcal{F}_{[m]} \simeq \mathcal{F}_{[n]}$.

$\Longleftarrow$ Assume that $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I)$ is satisfiable. Because $n \geq m$ holds, Proposition 4.24 yields that $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$ is satisfiable.

$\Longrightarrow$ Assume that $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$ is satisfiable. If $k = 1$, then $I = \emptyset$ and $m = 1$, and Theorem 4.8 yields correctness of Theorem 4.39. Assume $k \geq 2$. If $n \leq k - 1$, then $m = n$, and Theorem 4.39 holds. Assume $n > k - 1$. Then $m = k - 1$. Let $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$.

Then S satisfies I, and, according to Theorem 4.34, there exist $i_1, \ldots, i_{k-1} \in \{0, \ldots, n-1\}$ such that $S[i_1] \otimes \ldots \otimes S[i_{k-1}]$ satisfies I. Furthermore we have $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) = 0_{[n]}$, and thus, according to Theorem 4.9, $S[i_1] \otimes \ldots \otimes S[i_{k-1}]$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[k-1]})$. Consequently, $\mathbf{BvSAT}(\mathcal{F}_{[k-1]}, I) = \mathbf{BvSAT}(\mathcal{F}_{[m]}, I)$ is satisfiable. ■

Theorem 4.39 yields that, if the disequality graph $\mathcal{G}_{\neq}(\mathcal{F}_{[n]}, I)$ is connected, then satisfiability of a bitwise **BvSAT** problem for bitvectors of width n, induced by an arbitrary k-ary bitwise

bitvector function $\mathcal{F}_{[n]}$ and the disequality constraints of I, can be reduced to satisfiability of the corresponding bitwise **BvSAT** problem for bitvectors of width m while preserving satisfiability in a one-to-one fashion.

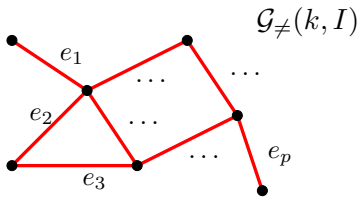
The next theorem yields that the amount of reduction down to m bits which is proposed in Theorem 4.39 is optimal in the sense that further reduction down to $< m$ bits violates a general one-to-one preservation of satisfiability (note that for $k \leq 2$ we have $m = 1$ which obviously already is the optimum reduction; furthermore note that for $k > 2$ and $n = k - 1$ we have $m = k - 1$):

Theorem 4.40 (Minimality) *Let $k \in \mathbb{N}_+$ with $k > 2$. Let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ such that the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$ is connected. Then there exists a family $(\mathcal{F}_{[i]})_{i \in \mathbb{N}_+}$ of corresponding k-ary bitwise bitvector functions such that*

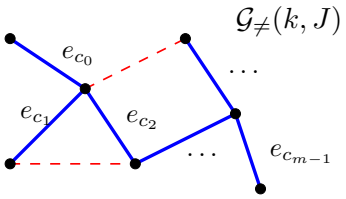
$$\textbf{BvSAT}(\mathcal{F}_{[k-1]}, I) \text{ is satisfiable} \quad \wedge \quad \textbf{BvSAT}(\mathcal{F}_{[k-2]}, I) \text{ is not satisfiable}$$

Proof: Let $k \in \mathbb{N}_+$ with $k > 2$. Let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ such that the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$ is connected. We will construct a satisfying solution of I which is of width $k - 1$, and from that we will construct a Boolean bitvector function which "accepts" (evaluates to zero) exactly all the columns of this satisfying solution. Then we show that the corresponding enhanced bitwise **BvSAT** problem is satisfiable for bitvector width $k - 1$, but not for width $k - 2$.

Let $p := |I|$ and let $e_1, \ldots, e_p \subseteq \{1, \ldots, k\}$ with $|e_i| = 2$ for all $i \in \{1, \ldots, p\}$, such that $I := \{e_1, \ldots, e_p\}$, as illustrated below.

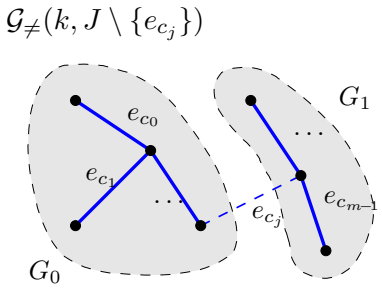


Let $m := k - 1$. Let $J \subseteq I$ such that $\mathcal{G}_{\neq}(k, J)$ is a spanning tree of $\mathcal{G}_{\neq}(k, I)$. Then $\mathcal{G}_{\neq}(k, J)$ is connected, and $|J| = m$. Let $c_0, \ldots, c_{m-1} \in \{1, \ldots, p\}$ such that $J = \{e_{c_0}, \ldots, e_{c_{m-1}}\}$.



For each $j \in \{0, \ldots, m - 1\}$, let $\boldsymbol{y}^{1,j}_{[1]}, \ldots, \boldsymbol{y}^{k,j}_{[1]} \in \mathbb{B}_{[1]}$ be defined in the following way:

After removing edge e_{c_j} from the spanning tree J, $\mathcal{G}_{\neq}(k, J \setminus \{e_{c_j}\})$ has exactly two connected components, say $G_0 \subseteq \{1, \ldots, k\}$ and $G_1 \subseteq \{1, \ldots, k\}$, as shown in the figure below.

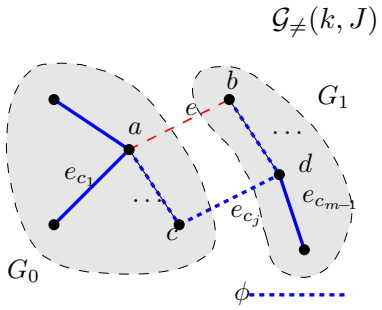


For each $i \in \{1, \ldots, k\}$ define

$$\boldsymbol{y}^{i,j}_{[1]} := \begin{cases} 0_{[1]} & \text{if} \quad i \in G_0 \\ 1_{[1]} & \text{if} \quad i \in G_1 \end{cases}$$

Then $\langle \boldsymbol{y}^{1,j}_{[1]}, \ldots, \boldsymbol{y}^{k,j}_{[1]} \rangle$ satisfies e_{c_j}, and moreover, any other disequality constraint $e' \in J \setminus e_{c_j}$ is not satisfied by $\langle \boldsymbol{y}^{1,j}_{[1]}, \ldots, \boldsymbol{y}^{k,j}_{[1]} \rangle$, i.e. $\langle \boldsymbol{y}^{1,j}_{[1]}, \ldots, \boldsymbol{y}^{k,j}_{[1]} \rangle$ satisfies exactly one disequality constraint of J. Let $\boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \in \mathbb{B}_{[m]}$ such that $\boldsymbol{x}^i_{[m]}[j] := \boldsymbol{y}^{i,j}_{[1]}$ for all $i \in \{1, \ldots, k\}$ and all $j \in \{0, \ldots, m-1\}$. Let $S := \langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle$. Then $S[j] = \langle \boldsymbol{y}^{1,j}_{[1]}, \ldots, \boldsymbol{y}^{k,j}_{[1]} \rangle$ for all $j \in \{0, \ldots, m-1\}$. Hence S satisfies J, since for all $j \in \{0, \ldots, m-1\}$ $S[j]$ satisfies e_{c_j}.

Now, let $e \in I \setminus J$, say $e = \{a, b\}$ for $a, b \in \{1, \ldots, k\}$. Because $\mathcal{G}_{\neq}(k, J)$ is a spanning tree of $\mathcal{G}_{\neq}(k, I)$ and thus connected, there exists a path ϕ in $\mathcal{G}_{\neq}(k, J)$ which leads from a to b. Let $j \in \{0, \ldots, m-1\}$ such that $e_{c_j} \in J$ is an edge on ϕ, say $e_{c_j} = \{c, d\}$ for $c, d \in \{1, \ldots, k\}$. Then, in $\mathcal{G}_{\neq}(k, J \setminus \{e_{c_j}\})$, a and b are *not* in the same connected component, as illustrated below.



Furthermore, c and d are in different connected components of $\mathcal{G}_{\neq}(k, J \setminus \{e_{c_j}\})$. Without loss of generality, we can assume that a is in the same component as c, and b is in the same component as d. By construction, $S[j] = \langle \boldsymbol{y}^{1,c_j}_{[1]}, \ldots, \boldsymbol{y}^{k,c_j}_{[1]} \rangle$ satisfies e_{c_j}, with $\boldsymbol{y}^{a,c_j}_{[1]} = \boldsymbol{y}^{c,c_j}_{[1]}$ and $\boldsymbol{y}^{b,c_j}_{[1]} = \boldsymbol{y}^{d,c_j}_{[1]}$. Then $\boldsymbol{y}^{c,c_j}_{[1]} \neq \boldsymbol{y}^{d,c_j}_{[1]}$ yields $\boldsymbol{y}^{a,c_j}_{[1]} \neq \boldsymbol{y}^{b,c_j}_{[1]}$ and thus $S[j]$ satisfies e.

Hence $S = \langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle$ satisfies all disequality constraints of I. Furthermore, all columns $S[i]$ and $S[j]$ of S with $i, j \in \{0, \ldots, m-1\}$ and $i \neq j$ are mutually different. We now construct a k-ary bitwise bitvector function $\mathcal{F}_{[m]}$ of width m, such that $\mathcal{F}_{[m]}(S') = \mathsf{0}_{[m]}$ if and only if S' is a composition of columns of S. Therefore, let $\mathcal{B}_{[1]} : \underbrace{\mathbb{B}_{[1]} \times \ldots \times \mathbb{B}_{[1]}}_{k} \longrightarrow \mathbb{B}_{[1]}$ be defined by:

$$\mathcal{B}_{[1]}(\boldsymbol{z}^1_{[1]}, \ldots, \boldsymbol{z}^k_{[1]}) \;:=\; \mathsf{neg}\Big(\underset{j \in \{0,\ldots,m-1\}}{\mathsf{OR}} \Big(\big(\underset{\substack{i \in \{1,\ldots,k\} \\ with \\ \boldsymbol{y}^{i,c_j}_{[1]} = \mathsf{1}_{[1]}}}{\mathsf{AND}} \boldsymbol{z}^j_{[1]} \big) \;\mathsf{and}\; \big(\underset{\substack{i \in \{1,\ldots,k\} \\ with \\ \boldsymbol{y}^{i,c_j}_{[1]} = \mathsf{0}_{[1]}}}{\mathsf{AND}} \mathsf{neg}(\boldsymbol{z}^j_{[1]}) \big) \Big) \Big)$$

for all $\boldsymbol{z}^1_{[1]}, \ldots, \boldsymbol{z}^k_{[1]} \in \mathbb{B}_{[1]}$. Then we have:

$$\begin{aligned} \mathcal{B}_{[1]}(\boldsymbol{z}^1_{[1]}, \ldots, \boldsymbol{z}^k_{[1]}) = \mathsf{0}_{[1]} \iff \exists\, j \in \{0, \ldots, m-1\} \;:\; \langle \boldsymbol{z}^1_{[1]}, \ldots, \boldsymbol{z}^k_{[1]} \rangle &= \langle \boldsymbol{y}^{1,c_j}_{[1]}, \ldots, \boldsymbol{y}^{k,c_j}_{[1]} \rangle \\ &= S[j] \end{aligned}$$

Let $\mathcal{F}_{[m]}$ be the k-ary bitwise bitvector function on bitvectors of width m with characteristic Boolean function $\mathcal{B}_{[1]}$, i.e. $\mathcal{F}_{[m]} \simeq \mathcal{B}_{[1]}$. Then we have:

$$\mathcal{F}_{[m]}(S') = \mathsf{0}_{[m]} \iff \forall\, j \in \{0, \ldots, m-1\}\; \exists\, i \in \{0, \ldots, m-1\} \;:\; S'[j] = S[i]$$

Hence, S is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I)$, i.e. $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I)$ is satisfiable, and each satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I)$ can be characterized as stated above.

We now show that $\mathbf{BvSAT}(\mathcal{F}_{[m-1]}, I)$ is unsatisfiable, where $\mathcal{F}_{[m-1]}$ denotes to corresponding k-ary bitwise bitvector function on bitvectors of width $m-1$ with $\mathcal{F}_{[m-1]} \simeq \mathcal{F}_{[m]}$. Let $\boldsymbol{z}^1_{[m-1]}, \ldots, \boldsymbol{z}^k_{[m-1]} \in \mathbb{B}_{[m-1]}$ such that $\mathcal{F}_{[m-1]}(\boldsymbol{z}^1_{[m-1]}, \ldots, \boldsymbol{z}^k_{[m-1]}) = \mathsf{0}_{[m-1]}$. Because $\mathcal{B}_{[1]}$ is also the characteristic Boolean function of $\mathcal{F}_{[m-1]}$, then $\langle \boldsymbol{z}^1_{[m-1]}, \ldots, \boldsymbol{z}^k_{[m-1]} \rangle$ is a composition of columns of $S = \langle \boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]} \rangle$.

Because $\boldsymbol{z}^1_{[m-1]}, \ldots, \boldsymbol{z}^k_{[m-1]}$ are bitvectors of width $m-1$, and because $\boldsymbol{x}^1_{[m]}, \ldots, \boldsymbol{x}^k_{[m]}$ are bitvectors of width m, there exists $j \in \{0, \ldots, m-1\}$ such that

$$\forall\, i \in \{0, \ldots, m-2\} \;:\; \langle \boldsymbol{z}^1_{[m-1]}, \ldots, \boldsymbol{z}^k_{[m-1]} \rangle[i] \;\neq\; S[j]$$

But then disequality constraint $e_{c_j} \in J$ is not satisfied by $\langle \boldsymbol{z}^1_{[m-1]}, \ldots, \boldsymbol{z}^k_{[m-1]} \rangle$. As the above reasoning holds for all $\langle \boldsymbol{z}^1_{[m-1]}, \ldots, \boldsymbol{z}^k_{[m-1]} \rangle$ with $\mathcal{F}_{[m-1]}(\boldsymbol{z}^1_{[m-1]}, \ldots, \boldsymbol{z}^k_{[m-1]}) = \mathsf{0}_{[m-1]}$, we conclude that $\mathbf{BvSAT}(\mathcal{F}_{[m-1]}, I)$ is unsatisfiable. ■

In Theorems 4.39 and 4.40 we have addressed the satisfiability problem for bitwise bitvector functions and bitvector disequalities while assuming the corresponding disequality graphs to be connected. Both theorems are now generalized for arbitrary instances of the satisfiability problem.

Theorem 4.41 (Satisfiability of BvSAT) *Let $k \in \mathbb{N}_+$ and $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Let $n \in \mathbb{N}_+$ and let*

$$m := \min(n, \max(1, k - p))$$

Then for each family $(\mathcal{F}_{[i]})_{i \in \mathbb{N}_+}$ of corresponding k-ary bitwise bitvector functions the following holds:

$$\mathsf{BvSAT}(\mathcal{F}_{[n]}, I) \text{ is satisfiable} \iff \mathsf{BvSAT}(\mathcal{F}_{[m]}, I) \text{ is satisfiable}$$

i.e. **BvSAT** *problems for bitvectors of width n, induced by an arbitrary set of disequality constraints I and an arbitrary bitwise bitvector function $\mathcal{F}_{[n]}$, can be reduced to the corresponding bitwise satisfiability problem for bitvectors of width m while preserving satisfiability in a one-to-one fashion.*

Proof: Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Let $n \in \mathbb{N}_+$ and let $\mathcal{F}_{[n]}$ be a k-ary bitwise bitvector function on bitvectors of width n. Let $m := \min(n, \max(1, k-p))$, and let $\mathcal{F}_{[m]}$ denote the corresponding k-ary bitwise bitvector function on bitvectors of width m with $\mathcal{F}_{[m]} \simeq \mathcal{F}_{[n]}$.

$\Longleftarrow$ Assume that $\mathsf{BvSAT}(\mathcal{F}_{[m]}, I)$ is satisfiable. Because $n \geq m$ holds, Proposition 4.24 yields that $\mathsf{BvSAT}(\mathcal{F}_{[n]}, I)$ is satisfiable.

$\Longrightarrow$ Assume that $\mathsf{BvSAT}(\mathcal{F}_{[n]}, I)$ is satisfiable. If $k = 1$, then $I = \emptyset$ and $m = 1$, and Theorem 4.8 yields correctness of Theorem 4.41. Assume $k \geq 2$. If $n \leq k - 1$, then $m = n$, and Theorem 4.41 holds. Assume $n > k - 1$. Then $m = k - 1$. Let $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ is a satisfying solution of $\mathsf{BvSAT}(\mathcal{F}_{[n]}, I)$.

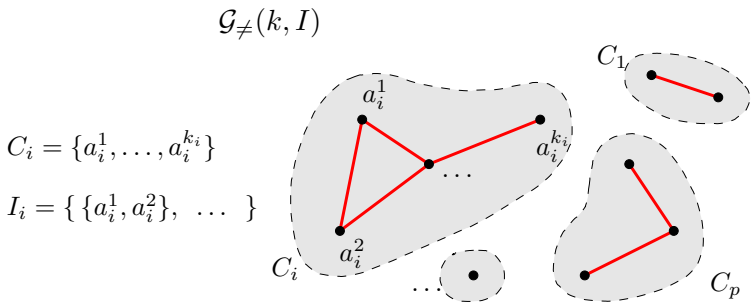
Then S satisfies I, and according to Theorem 4.35 there exist $i_1, \ldots, i_{k-p} \in \{0, \ldots, n-1\}$ such that $S[i_1] \otimes \ldots \otimes S[i_{k-p}]$ satisfies I. Furthermore, we have $\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]}) = 0_{[n]}$, and thus, according to Theorem 4.9, $S[i_1] \otimes \ldots \otimes S[i_{k-p}]$ is a satisfying solution of $\mathsf{BvSAT}(\mathcal{F}_{[k-p]})$. Consequently, $\mathsf{BvSAT}(\mathcal{F}_{[k-p]}, I) = \mathsf{BvSAT}(\mathcal{F}_{[m]}, I)$ is satisfiable. ■

Theorem 4.42 (Minimality) *Let $k \in \mathbb{N}_+$ with $k > 2$. Let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Then there exists a family $(\mathcal{F}_{[i]})_{i \in \mathbb{N}_+}$ of corresponding k-ary bitwise bitvector functions such that*

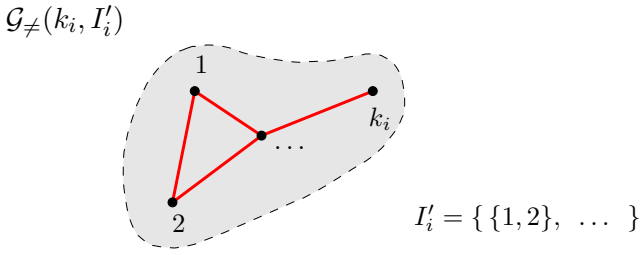
$$\mathsf{BvSAT}(\mathcal{F}_{[k-p]}, I) \text{ is satisfiable} \quad \wedge \quad \mathsf{BvSAT}(\mathcal{F}_{[k-p-1]}, I) \text{ is not satisfiable}$$

Proof: Let $k \in \mathbb{N}_+$ with $k > 2$. Let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph, and let $m := k - p$.

Let $C_1, \ldots, C_p \subseteq \{1, \ldots, k\}$ be the connected components of $\mathcal{G}_{\neq}(k, I)$. For each $i \in \{1, \ldots, p\}$ let $k_i := |C_i|$ and let $a_i^1, \ldots, a_i^{k_i} \in \{1, \ldots, k\}$ such that $C_i = \{a_i^1, \ldots, a_i^{k_i}\}$. Furthermore, let $I_i := \{\, e \in I \mid e \subseteq C_i \,\}$ be the set of disequality constraints related to component C_i.



Let $I_i' := \{\, \{\alpha, \beta\} \mid \alpha, \beta \in \{1, \ldots, k_i\} \wedge \{a_i^\alpha, a_i^\beta\} \in I_i \,\}$. Then $\mathcal{G}_{\neq}(k_i, I_i')$ is connected.



Let $m_i := k_i - 1$. According to Theorem 4.40, there exists a k_i-ary bitwise bitvector function $\mathcal{F}^i_{[m_i]}$ such that $\mathbf{BvSAT}(\mathcal{F}^i_{[m_i]}, I_i')$ is satisfiable and $\mathbf{BvSAT}(\mathcal{F}^i_{[m_i-1]}, I_i')$ is not satisfiable. For each $i \in \{1, \ldots, p\}$ let $\mathcal{B}^i_{[1]}$ be the characteristic Boolean function of the respective $\mathcal{F}^i_{[m_i]}$, and let

$$\mathcal{B}_{[1]} : \underbrace{\mathbb{B}_{[1]} \times \ldots \times \mathbb{B}_{[1]}}_{k} \longrightarrow \mathbb{B}_{[1]}$$

be defined by

$$\mathcal{B}_{[1]}(\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]}) \quad := \quad \begin{cases} 0_{[1]} & \textit{if} \quad \exists\, i \in \{1, \ldots, p\} \ :\ \mathcal{B}^i_{[1]}(\boldsymbol{x}^{a_i^1}_{[1]}, \ldots, \boldsymbol{x}^{a_i^{k_i}}_{[1]}) = 0_{[1]} \ \wedge \\ & \qquad \forall\, j \in \{1, \ldots, p\} \setminus \{i\}\ \forall\, l \in \{1, \ldots, k_j\} \ :\ \boldsymbol{x}^{a_j^l}_{[1]} = 0_{[1]} \\ 1_{[1]} & \textit{else} \end{cases}$$

for all $\boldsymbol{x}^1_{[1]}, \ldots, \boldsymbol{x}^k_{[1]} \in \mathbb{B}_{[1]}$ (an illustration of the construction is given on the next page), and let $\mathcal{F}_{[m]}$ be the corresponding k-ary bitwise bitvector function with characteristic Boolean function $\mathcal{B}_{[1]}$, i.e. $\mathcal{F}_{[m]} \simeq \mathcal{B}_{[1]}$. We now show that $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I)$ is satisfiable and that $\mathbf{BvSAT}(\mathcal{F}_{[m-1]}, I)$ is not satisfiable.

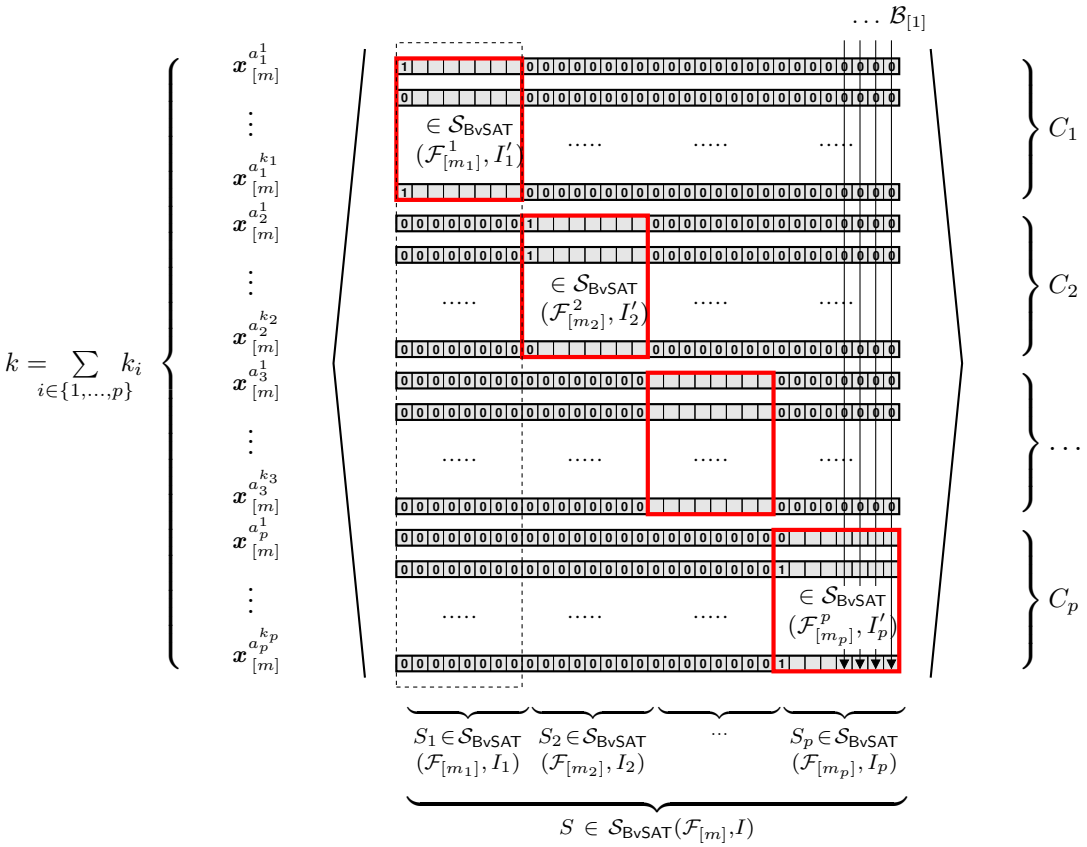
For each $i \in \{1, \ldots, p\}$, let $\boldsymbol{x}^{i,1}_{[m_i]}, \ldots, \boldsymbol{x}^{i,k_i}_{[m_i]} \in \mathbb{B}_{[m_i]}$ such that $\langle \boldsymbol{x}^{i,1}_{[m_i]}, \ldots, \boldsymbol{x}^{i,k_i}_{[m_i]} \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}^i_{[m_i]}, I'_i)$. Then all columns of $\langle \boldsymbol{x}^{i,1}_{[m_i]}, \ldots, \boldsymbol{x}^{i,k_i}_{[m_i]} \rangle$ are pairwise distinct from each other, and $\mathbf{BvSAT}(\mathcal{F}^i_{[m_i-1]}, I'_i)$ is not satisfiable. Let $\boldsymbol{y}^1_{[m_i]}, \ldots, \boldsymbol{y}^k_{[m_i]} \in \mathbb{B}_{[m_i]}$ with

$$\boldsymbol{y}^j_{[m_i]} := \begin{cases} \boldsymbol{x}^{i,l}_{[m_i]} & \text{if} \quad \exists\, l \in \{1, \ldots, k_i\} \;:\; a^l_i = j \\ 0_{[m_i]} & \text{else} \end{cases}$$

Let $S_i := \langle \boldsymbol{y}^1_{[m_i]}, \ldots, \boldsymbol{y}^k_{[m_i]} \rangle$. Then S_i is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[m_i]}, I_i)$, and for each $e \in I \setminus I_i$ we have that S_i does not satisfy e. Now define $S := S_1 \otimes \ldots \otimes S_p$. Then S is of width m, because

$$\sum_{i \in \{1,\ldots,p\}} m_i \;=\; \sum_{i \in \{1,\ldots,p\}} (k_i - 1) \;=\; k - p \;=\; m\,,$$

and by construction S is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I)$. Hence, $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I)$ is satisfiable. The construction of S is illustrated below.



Due to the definition of $\mathcal{B}_{[1]}$, the satisfiability problem $\mathbf{BvSAT}(\mathcal{B}_{[1]})$ has exactly m satisfying solutions. As S is of width m, and as all columns of S are pairwise distinct from each other, the set of satisfying solutions of $\mathbf{BvSAT}(\mathcal{B}_{[1]})$ consists exactly of the columns of S.

Assume that $\mathbf{BvSAT}(\mathcal{F}_{[m-1]}, I)$ is satisfiable, and let S' be a satisfying solution. Because $\mathcal{F}_{[m-1]} \simeq \mathcal{B}_{[1]}$, then S' is a composition of $m-1$ columns of S, and as all columns of S are distinct, there must exist $j \in \{0, \ldots, m-1\}$ such that $S[j]$ is different from all columns of S'.

But then there exists $i \in \{1, \ldots, p\}$ for which $\mathbf{BvSAT}(\mathcal{F}_{[m_i-1]}, I_i)$ is satisfiable and for which thus $\mathbf{BvSAT}(\mathcal{F}^i_{[m_i-1]}, I'_i)$ is also satisfiable. This is contradictory to the selection of $\mathcal{F}^i_{[m_i]}$ according to Theorem 4.40. $\lightning$ ■

From Theorem 4.41 and Theorem 4.42 we can now conclude that the global satisfiability threshold for bitwise bitvector functions and sets of disequality constraints is efficiently computable.

Theorem 4.43 (Efficient Computability of $\nu(k, I)$) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Then we have*

$$\nu(k, I) \;=\; \max(1, k - p)$$

Proof: Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ be a set of disequality constraints. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Define:

$$q := \max(1, k - p)$$

Let $n \in \mathbb{N}_+$ and let $\mathcal{F}_{[n]}$ be a k-ary bitwise bitvector function on bitvectors of width n, such that $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I)$ is satisfiable, and let $m := \min(n, \max(1, k - p))$. Then Theorem 4.41 yields that $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I)$ is also satisfiable. Since $m \leq q$, we can conclude $\mu(\mathcal{F}_{[n]}, I) \leq q$, and therefore

$$\nu(k, I) \leq q$$

Furthermore, according to Theorem 4.42, there exists a family $(\mathcal{F}_{[i]})_{i \in \mathbb{N}_+}$ of k-ary bitwise bitvector functions such that $\mathbf{BvSAT}(\mathcal{F}_{[q]}, I)$ is satisfiable, and $\mathbf{BvSAT}(\mathcal{F}_{[q-1]}, I)$ is not satisfiable. Then $\mu(\mathcal{F}_{[m]}, I) = q$, and thus

$$\nu(k, I) \geq q$$

Hence, we have $\nu(k, I) = q = \max(1, k - p)$. ■

We conclude this chapter by presenting a further generalization of Theorem 4.41. Given a set I of disequality constraints, we characterize satisfiability of **BvSAT** for bitwise bitvector functions and for arbitrary subsets I' of I.

Theorem 4.44 (Main Theorem) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Let $n \in \mathbb{N}_+$ and let*

$$m := \min(n, \max(1, k - p))$$

Then for all k-ary bitwise bitvector functions $\mathcal{F}_{[n]}$ and all $I' \subseteq I$ the following holds:

$$\begin{array}{c} \mathsf{BvSAT}(\mathcal{F}_{[n]}, I') \\ \textit{is satisfiable} \end{array} \iff \begin{array}{c} \mathsf{BvSAT}(\mathcal{F}_{[m]}, I') \\ \textit{is satisfiable} \end{array}$$

Proof: Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Let $n \in \mathbb{N}_+$ and let $m := \min(n, \max(1, k - p))$. Let $\mathcal{F}_{[n]}$ be a k-ary bitwise bitvector function of width n. Let $\mathcal{F}_{[m]}$ denote the corresponding bitwise bitvector function of width m with $\mathcal{F}_{[m]} \simeq \mathcal{F}_{[n]}$ and let $I' \subseteq I$. According to Proposition 4.24 we then have:

$$\mathsf{BvSAT}(\mathcal{F}_{[m]}, I') \implies \mathsf{BvSAT}(\mathcal{F}_{[n]}, I')$$

Let $p' \in \mathbb{N}_+$ be the number of connected components of the disequality graph $\mathcal{G}_{\neq}(k, I')$. Because of $I' \subseteq I$ we then have $p' \geq p$. Let $m' := \min(n, \max(1, k - p'))$. Then Theorem 4.41 yields:

$$\mathsf{BvSAT}(\mathcal{F}_{[n]}, I') \iff \mathsf{BvSAT}(\mathcal{F}_{[m']}, I') \tag{4.1}$$

Furthermore we have:

$$\begin{array}{lll} p' \geq p & \implies & k - p \geq k - p' \\ & \implies & \max(1, k - p) \geq \max(1, k - p') \\ & \implies & \min(n, \max(1, k - p)) \geq \min(n, \max(1, k - p')) \\ & \implies & m \geq m' \end{array}$$

and thus according to Proposition 4.24

$$\mathsf{BvSAT}(\mathcal{F}_{[m']}, I') \implies \mathsf{BvSAT}(\mathcal{F}_{[m]}, I') \tag{4.2}$$

Then (4.1) and (4.2) yield

$$\mathsf{BvSAT}(\mathcal{F}_{[n]}, I') \implies \mathsf{BvSAT}(\mathcal{F}_{[m]}, I')$$

and hence Theorem 4.44 holds. ■

4.9 Order Constraints

In this section, the previous results are further refined by showing that all reduction theorems which have been presented so far are still valid if the requirement of disequality of bitvectors in the enhanced **BvSAT** problem is replaced by strict inequality of bitvectors.

Definition 4.45 (Strict Inequality) *Let $n \in \mathbb{N}_+$. **Strict inequality** of fixed-size bitvectors of width n is defined as follows:*

$$\boldsymbol{x}_{[n]} < \boldsymbol{y}_{[n]} \quad :\Longleftrightarrow \quad \mathsf{bv2nat}(\boldsymbol{x}_{[n]}) < \mathsf{bv2nat}(\boldsymbol{y}_{[n]})$$

for $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. ■

Strict inequality can be characterized in the following way:

Proposition 4.46 (Strict Inequality) *Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. Then the following holds:*

$$\begin{aligned} \boldsymbol{x}_{[n]} < \boldsymbol{y}_{[n]} \quad \Longleftrightarrow \quad & \exists\, i \in \{0,\ldots,n-1\} \;:\; (\, \boldsymbol{x}_{[n]}[i] < \boldsymbol{y}_{[n]}[i] \,) \;\wedge \\ & \forall\, j \in \{0,\ldots,n-1\} \;:\; (\, j \geq i \implies \boldsymbol{x}_{[n]}[j] \leq \boldsymbol{y}_{[n]}[j] \,) \end{aligned}$$

Proof: $\Longrightarrow$ Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$ with $\boldsymbol{x}_{[n]} < \boldsymbol{y}_{[n]}$. According to Definition 4.45, we have $\mathsf{bv2nat}(\boldsymbol{x}_{[n]}) < \mathsf{bv2nat}(\boldsymbol{y}_{[n]})$ and thus $\boldsymbol{x}_{[n]} \neq \boldsymbol{y}_{[n]}$. Let

$$c \;:=\; \max\{\, i \in \{0,\ldots,n-1\} \mid \boldsymbol{x}_{[n]}[i] \neq \boldsymbol{y}_{[n]}[i] \,\}. \tag{4.3}$$

Then $\boldsymbol{x}_{[n]}[n{-}1, c{+}1] = \boldsymbol{y}_{[n]}[n{-}1, c{+}1]$, $\boldsymbol{x}_{[n]}[c] \neq \boldsymbol{y}_{[n]}[c]$ and $\mathsf{bv2nat}(\boldsymbol{x}_{[n]}[c,0]) < \mathsf{bv2nat}(\boldsymbol{y}_{[n]}[c,0])$. Assume $\boldsymbol{x}_{[n]}[c] = \mathsf{1}_{[1]}$ and $\boldsymbol{y}_{[n]}[c] = \mathsf{0}_{[1]}$. Then

$$\begin{aligned} & \mathsf{bv2nat}(\boldsymbol{x}_{[n]}[c,0]) && < && \mathsf{bv2nat}(\boldsymbol{y}_{[n]}[c,0]) \\ \Longrightarrow \quad & \sum_{0\leq i\leq c} \boldsymbol{x}_{[n]}[i] \cdot 2^i && < && \sum_{0\leq i\leq c} \boldsymbol{y}_{[n]}[i] \cdot 2^i \\ \Longrightarrow \quad & \boldsymbol{x}_{[n]}[c] \cdot 2^c + \sum_{0\leq i<c} \boldsymbol{x}_{[n]}[i] \cdot 2^i && < && \boldsymbol{y}_{[n]}[c] \cdot 2^c + \sum_{0\leq i<c} \boldsymbol{y}_{[n]}[i] \cdot 2^i \\ \Longrightarrow \quad & 2^c + \sum_{0\leq i<c} \boldsymbol{x}_{[n]}[i] \cdot 2^i && < && 0 + \sum_{0\leq i<c} \boldsymbol{y}_{[n]}[i] \cdot 2^i \\ \Longrightarrow \quad & 2^c && < && \sum_{0\leq i<c} 1 \cdot 2^i \\ \Longrightarrow \quad & 2^c && < && 2^c - 1 \quad \lightning \end{aligned}$$

Hence, we have $\boldsymbol{x}_{[n]}[c] = \mathsf{0}_{[1]}$ and $\boldsymbol{y}_{[n]}[c] = \mathsf{1}_{[1]}$ and thus $\boldsymbol{x}_{[n]}[c] < \boldsymbol{y}_{[n]}[c]$, and, according to (4.3), for all $j \in \{c+1,\ldots,n-1\}$ we have $\boldsymbol{x}_{[n]}[j] = \boldsymbol{y}_{[n]}[j]$ and thus $\boldsymbol{x}_{[n]}[j] \leq \boldsymbol{y}_{[n]}[j]$.

$\Longleftarrow$ Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]} \in \mathbb{B}_{[n]}$. Let $c \in \{0, \ldots, n-1\}$ with $\boldsymbol{x}_{[n]}[c] < \boldsymbol{y}_{[n]}[c]$, and for all $j \in \{c+1, \ldots, n-1\}$ let $\boldsymbol{x}_{[n]}[j] \leq \boldsymbol{y}_{[n]}[j]$. Then $\boldsymbol{x}_{[n]}[c] = 0_{[0]}$, $\boldsymbol{y}_{[n]}[c] = 1_{[1]}$, and $\boldsymbol{x}_{[n]}[n-1, c+1] \leq \boldsymbol{y}_{[n]}[n-1, c+1]$, and we have:

$$
\begin{array}{lrcl}
 & \mathsf{bv2nat}(\boldsymbol{x}_{[n]}[n-1, c+1]) & \leq & \mathsf{bv2nat}(\boldsymbol{y}_{[n]}[n-1, c+1]) \\
\Longrightarrow & \sum_{c<i\leq n-1} \boldsymbol{x}_{[n]}[i] \cdot 2^i & \leq & \sum_{c<i\leq n-1} \boldsymbol{y}_{[n]}[i] \cdot 2^i \\
\Longrightarrow & \sum_{0\leq i<c-1} \boldsymbol{x}_{[n]}[i] \cdot 2^i + \sum_{c<i<n-1} \boldsymbol{x}_{[n]}[i] \cdot 2^i & < & 2^c + \sum_{0\leq i<c-1} \boldsymbol{y}_{[n]}[i] \cdot 2^i \\
\Longrightarrow & \sum_{0\leq i<n-1} \boldsymbol{x}_{[n]}[i] \cdot 2^i & < & 2^c + \sum_{0\leq i<c-1} \boldsymbol{y}_{[n]}[i] \cdot 2^i \\
\Longrightarrow & \mathsf{bv2nat}(\boldsymbol{x}_{[n]}) & < & \mathsf{bv2nat}(\boldsymbol{y}_{[n]}[n-1, c]) \\
\Longrightarrow & \boldsymbol{x}_{[n]} & < & \boldsymbol{y}_{[n]}
\end{array}
$$

■

Strict inequalities are also referred to as *order constraints*. Satisfying solutions of disequality constraints can be reduced while preserving order constraints.

Theorem 4.47 (Order Constraints and Disequality Graphs) *Let $k \in \mathbb{N}_+$ and $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ with $I \neq \emptyset$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ satisfies I. Let $m := \max(1, k-p)$. Then there exist columns $j_1, \ldots, j_m \in \{0, \ldots, n-1\}$ and bitvectors $\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^k_{[m]} \in \mathbb{B}_{[m]}$ such that*

$$S' \;=\; \langle \boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^k_{[m]} \rangle \;:=\; S[j_1] \;\otimes\; \ldots \;\otimes\; S[j_m]$$

is a composition of columns of S which satisfies I and for which the following holds:

$$\forall\, \{i, j\} \in I \;:\; (\boldsymbol{x}^i_{[n]} > \boldsymbol{x}^j_{[n]} \implies \boldsymbol{y}^i_{[m]} > \boldsymbol{y}^j_{[m]}) \;\wedge\; (\boldsymbol{x}^i_{[n]} < \boldsymbol{x}^j_{[n]} \implies \boldsymbol{y}^i_{[m]} < \boldsymbol{y}^j_{[m]})$$

i.e. there exists a satisfying solution S' of I which is of width m and which preserves all order constraints of I in the same way as they are satisfied by S.

Proof: Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$ with $I \neq \emptyset$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Let $n \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ satisfies I.

Let $q := |I|$ and let $e_1, \ldots, e_q \subseteq \{1, \ldots, k\}$ with $|e_i| = 2$ for all $i \in \{1, \ldots, q\}$, such that $I = \{e_1, \ldots, e_q\}$. According to Lemma 4.17, there exist columns $c_1, \ldots, c_q \in \{0, \ldots, n-1\}$ such that for each $i \in \{1, \ldots, q\}$, disequality constraint e_i is satisfied by $S[c_i]$. Without loss of generality, let each $c_1, \ldots, c_q$ be chosen as

$$c_i \;:=\; \max \{\; j \in \{0, \ldots, n-1\} \;\mid\; S[j] \textit{ satisfies } e_i \;\} \tag{4.4}$$

for each $i \in \{1, \ldots, q\}$, i.e. each $S[c_i]$ is the leftmost column of S which satisfies disequality constraint e_i. Let $\sigma : \{0, \ldots, q-1\} \longrightarrow \{1, \ldots, q\}$ be a bijection with

$$c_{\sigma(0)} \;\leq\; c_{\sigma(1)} \;\leq\; \ldots \;\leq\; c_{\sigma(q-1)} \tag{4.5}$$

i.e. σ enumerates $c_1, \ldots, c_q$ in increasing order. Then for each $i \in \{0, \ldots, q-1\}$ column $S[c_{\sigma(i)}]$ satisfies disequality constraint $e_{\sigma(i)}$. We construct a selection of $k-p$ columns of S by determining $j_0, \ldots, j_{k-p-1} \in \{0, \ldots, q-1\}$ such that $S[c_{\sigma(j_{k-p-1})}] \otimes \ldots \otimes S[c_{\sigma(j_1)}] \otimes S[c_{\sigma(j_0)}]$ satisfies I. Therefore, initially let $J := I$ and $i := q-1$ and $b := k-p$. Then $j_0, \ldots, j_{k-p-1}$ are computed in the following way:

1. If $b = 0$ then stop.
2. If $e_{\sigma(i)} \notin J$ then continue with step 4, otherwise:
   (a) Decrease b by one, i.e. let $b := b-1$, and let $j_b := i$. Then $S[c_{\sigma(j_b)}]$ satisfies $e_{\sigma(j_b)}$.
   (b) If $\mathcal{G}_{\neq}(k, J \setminus \{e_{\sigma(i)}\})$ has more connected components than $\mathcal{G}_{\neq}(k, J)$, then continue with step 3. Otherwise $e_{\sigma(i)}$ lies on a cycle, say γ, in $\mathcal{G}_{\neq}(k, J)$.
   (c) According to Theorem 4.32 then there exists $e' \in J$ on the same cycle γ with $e' \neq e_{\sigma(i)}$ such that $S[c_{\sigma(j_b)}]$ also satisfies e'. Remove e' from J, i.e. let $J := J \setminus \{e'\}$, and repeat step 2(b).
3. Remove $e_{\sigma(i)}$ from J, i.e. let $J := J \setminus \{e_{\sigma(i)}\}$.
4. Decrease i by one, i.e. let $i := i-1$, and continue with step 1.

The procedure given above determines $j_0, \ldots, j_{k-p-1}$ such that $e_{\sigma(j_0)}, \ldots, e_{\sigma(j_{k-p-1})}$ form a spanning tree of $\mathcal{G}_{\neq}(k, I)$. This is done by successively destroying cycles and removing edges from $\mathcal{G}_{\neq}(k, J)$. The initial value $k-p$ of b is decreased whenever an edge is removed which does not lie on any cycle. In that case, the number of connected components of $\mathcal{G}_{\neq}(k, J)$ increases. The maximum number of connected components, which can be reached, is k, and the initial number of connected components is p. Thus b is decreased exactly $k-p$ times, i.e. finally $b = 0$ holds and the procedure terminates. Furthermore we have:

$$j_0 \;<\; j_1 \;<\; \ldots \;<\; j_{k-p-1} \tag{4.6}$$

and thus according to (4.5) we conclude:

$$c_{\sigma(j_0)} \;\leq\; c_{\sigma(j_1)} \;\leq\; \ldots \;\leq\; c_{\sigma(j_{k-p-1})} \tag{4.7}$$

Let

$$S' \;:=\; S[c_{\sigma(j_{k-p-1})}] \otimes \ldots \otimes S[c_{\sigma(j_1)}] \otimes S[c_{\sigma(j_0)}]$$

and let $\boldsymbol{y}^1_{[k-p]}, \ldots, \boldsymbol{y}^k_{[k-p]} \in \mathbb{B}_{[k-p]}$ such that $S' = \langle\!\langle \boldsymbol{y}^1_{[k-p]}, \ldots, \boldsymbol{y}^k_{[k-p]} \rangle\!\rangle$. Then S' satisfies I (cf. proofs of Theorems 4.34 and 4.35 in Section 4.7).

Now, let $l \in \{0, \ldots, q-1\}$ and let $i, j \in \{1, \ldots, k\}$ such that $e_{\sigma(l)} = \{i, j\}$. Assume that S satisfies $e_{\sigma(l)}$ with $\boldsymbol{x}^i_{[n]} < \boldsymbol{x}^j_{[n]}$. Then according to (4.4) and Proposition 4.46 we have:

$$\left(\boldsymbol{x}^i_{[n]}[c_{\sigma(l)}] < \boldsymbol{x}^j_{[n]}[c_{\sigma(l)}] \right) \;\wedge\; \forall\, t \in \{0, \ldots, n-1\} \;:\; \left(t \geq c_{\sigma(l)} \implies \boldsymbol{x}^i_{[n]}[t] \leq \boldsymbol{x}^j_{[n]}[t] \right) \quad (4.8)$$

Case 1 : Edge $e_{\sigma(l)}$ was removed from J in step 3 of the above given procedure. Then there exists $w \in \{0, \ldots, k-p-1\}$ such that $l = j_w$. Then $S[c_{\sigma(j_w)}]$ satisfies $e_{\sigma(l)}$, and because of

$$S'[w] \;=\; S[c_{\sigma(j_w)}] \;=\; S[c_{\sigma(l)}]$$

then $S'[w]$ satisfies $e_{\sigma(l)}$. Furthermore, we then have:

$$\boldsymbol{y}^i_{[k-p]}[w] = \boldsymbol{x}^i_{[n]}[c_\sigma(l)] \quad \wedge \quad \boldsymbol{y}^j_{[k-p]}[w] = \boldsymbol{x}^j_{[n]}[c_\sigma(l)]$$

and (4.8) yields $\boldsymbol{y}^i_{[q]}[l] < \boldsymbol{y}^j_{[q]}[l]$. Now, let $v \in \{0, \ldots, k-p-1\}$ with $v \geq w$. Then (4.6), (4.5) and (4.8) yield:

$$\begin{aligned} v \geq w \quad &\implies \quad j_v \geq j_w \\ &\implies \quad c_{\sigma(j_v)} \geq c_{\sigma(j_w)} \\ &\implies \quad c_{\sigma(j_v)} \geq c_{\sigma(l)} \\ &\implies \quad \boldsymbol{x}^i_{[n]}[c_{\sigma(j_v)}] \leq \boldsymbol{x}^j_{[n]}[c_{\sigma(j_v)}] \\ &\implies \quad \boldsymbol{y}^i_{[k-p]}[v] \leq \boldsymbol{y}^j_{[k-p]}[v] \end{aligned}$$

Proposition 4.46 then yields $\boldsymbol{y}^i_{[k-p]} < \boldsymbol{y}^j_{[k-p]}$.

Case 2 : Edge $e_{\sigma(l)}$ was removed from J in step 2(c) of the above given procedure. Then there exist $l' \in \{0, \ldots, q-1\}$ and $w \in \{0, \ldots, k-p-1\}$ such that $e_{\sigma(l')}$ was selected earlier in step 2 and removed from J in step 3 and $l' = j_w$ and $S[c_{\sigma(j_w)}]$ satisfies both $e_{\sigma(l')}$ and $e_{\sigma(l)}$. Because of (4.4) then

$$c_{\sigma(l)} \;\geq\; c_{\sigma(j_w)} \;=\; c_{\sigma(l')}$$

Furthermore $l < l'$, and thus due to (4.5)

$$c_{\sigma(l)} \;\leq\; c_{\sigma(l')} \;=\; c_{\sigma(j_w)}$$

and hence $c_{\sigma(l)} \;=\; c_{\sigma(l')} \;=\; c_{\sigma(j_w)}$. Then

$$S'[w] \;=\; S[c_{\sigma(j_w)}] \;=\; S[c_{\sigma(l)}]$$

and $S'[w]$ satisfies $e_{\sigma(l)}$. Furthermore, we then have:

$$\boldsymbol{y}^i_{[k-p]}[w] = \boldsymbol{x}^i_{[n]}[c_\sigma(l)] \quad \wedge \quad \boldsymbol{y}^j_{[k-p]}[w] = \boldsymbol{x}^j_{[n]}[c_\sigma(l)]$$

and (4.8) yields $\boldsymbol{y}^i_{[q]}[l] < \boldsymbol{y}^j_{[q]}[l]$. Now, let $v \in \{0, \ldots, k-p-1\}$ with $v \geq w$. Then (4.6), (4.5) and (4.8) yield:

$$\begin{aligned} v \geq w \quad &\Longrightarrow \quad j_v \geq j_w \\ &\Longrightarrow \quad c_{\sigma(j_v)} \geq c_{\sigma(j_w)} \\ &\Longrightarrow \quad c_{\sigma(j_v)} \geq c_{\sigma(l)} \\ &\Longrightarrow \quad \boldsymbol{x}^i_{[n]}[c_{\sigma(j_v)}] \leq \boldsymbol{x}^j_{[n]}[c_{\sigma(j_v)}] \\ &\Longrightarrow \quad \boldsymbol{y}^i_{[k-p]}[v] \leq \boldsymbol{y}^j_{[k-p]}[v] \end{aligned}$$

Proposition 4.46 then yields $\boldsymbol{y}^i_{[k-p]} < \boldsymbol{y}^j_{[k-p]}$.

As l and thus $e_{\sigma(l)} = \{i, j\}$ was chosen arbitrarily, and as σ is bijective, we have shown:

$$\forall\ \{i,j\} \in I\ :\ \boldsymbol{x}^i_{[n]} < \boldsymbol{x}^j_{[n]} \iff \boldsymbol{y}^i_{[k-p]} < \boldsymbol{y}^j_{[k-p]}$$

Thus, S' is a satisfying solution of I which exactly preserves all order constraints of I which are satisfied by S. ■

As a result, Theorem 4.44 can be refined in the following way:

Theorem 4.48 (Order Constraints) *Let $k \in \mathbb{N}_+$ and let $I \subseteq \{\, e \subseteq \{1, \ldots, k\} \mid |e| = 2 \,\}$. Let $p \in \mathbb{N}_+$ be the number of connected components of the corresponding disequality graph $\mathcal{G}_{\neq}(k, I)$. Let $n \in \mathbb{N}_+$ and let*

$$m := \min(n, \max(1, k-p))$$

Let $I' \subseteq I$ and let $(\mathcal{F}_{[i]})_{i \in \mathbb{N}_+}$ be a family of corresponding k-ary bitwise bitvector functions. Assume that $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I')$ is satisfiable, and let $\boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ such that $S := \langle \boldsymbol{x}^1_{[n]}, \ldots, \boldsymbol{x}^k_{[n]} \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[n]}, I')$.

Then there exist $\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^k_{[m]} \in \mathbb{B}_{[m]}$ such that $S' := \langle \boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^k_{[m]} \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I')$ and the following holds:

$$\forall\ \{i,j\} \in I'\ :\ \left(\boldsymbol{x}^i_{[n]} > \boldsymbol{x}^j_{[n]} \implies \boldsymbol{y}^i_{[m]} > \boldsymbol{y}^j_{[m]}\right) \ \wedge\ \left(\boldsymbol{x}^i_{[n]} < \boldsymbol{x}^j_{[n]} \implies \boldsymbol{y}^i_{[m]} < \boldsymbol{y}^j_{[m]}\right)$$

i.e. there exists a satisfying solution of $\mathbf{BvSAT}(\mathcal{F}_{[m]}, I')$ which strictly preserves all order constraints of I' which are satisfied by S.

Proof: Follows from Theorem 4.44 and Theorem 4.47. ■

Chapter 5

Bitvector Terms and Systems of Bitvector Equations

In this chapter, a formal language of *bitvector terms* is introduced which is used throughout this thesis to specify Bounded Model Checking Problems on Register-Transfer-Level. We define bitvector terms and *systems of equations of bitvector terms* and present the notion of *satisfiability* and of *satisfying solutions of bitvector equations*. The expressiveness of the presented bitvector language is strong enough to describe digital circuit behavior in a way such that functional properties of a design can be verified by checking satisfiability of appropriate bitvector equations.

In addition, we show that satisfiability of bitvector equations can be characterized by **BvSAT** problems. The semantics of bitvector terms is given by defining how bitvector terms are interpreted by bitvector functions. Thus, bitvector terms provide a means of textual representation of bitvectors and bitvector functions, and therefore can be used to describe **BvSAT** problems.

5.1 Bitvector Terms

Let $k \in \mathbb{N}_+$ and let V be a finite set of k variable symbols for typed (fixed-size) bitvector variables. Throughout this thesis, we will implicitly assume that there is always given a strict and fixed ordering on the variable symbols, i.e. there is given a bijective function $\omega : \{1, \ldots, k\} \longrightarrow V$ which enumerates the symbols of V. Mostly, ω will not be stated explicitly, but will be obvious from the context, as, for example, in the case of $V = \{\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{c}_{[8]}, \boldsymbol{d}_{[8]}, \ldots, \boldsymbol{z}_{[8]}\}$ where ω is assumed to be the lexicographical ordering of the variables symbols. In general, we will denote finite ordered sets of bitvector variables as follows: for a given $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$ let

$$V := \{\boldsymbol{x}^1_{[n_1]}, \boldsymbol{x}^2_{[n_2]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}\,.$$

Then the ordering of the variable symbols is implicitly assumed to be given by $\omega(i) := \boldsymbol{x}^i_{[n_i]}$ for all $i \in \{1, \ldots, k\}$. The set $\mathcal{T}(V)$ of well-formed bitvector terms over a set V of bitvector variables is defined inductively over the bitvector variable symbols of V and a set of operator symbols for operations on fixed-size bitvectors. This set includes operator symbols for all standard operators on bitvectors which have been defined in the prior section.

Definition 5.1 (Bitvector Terms and Widths of Bitvector Terms) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of k bitvector variables of widths $n_1, \ldots, n_k$. The set $\mathcal{T}(V)$ of **well-formed bitvector terms** over variables of V and the **width of a bitvector term** $t \in \mathcal{T}(V)$ are defined inductively in the following way:*

- *For each $i \in \{1, \ldots, k\}$, let $\boldsymbol{x}^i_{[n_i]} \in \mathcal{T}(V)$ and $\mathsf{width}(\boldsymbol{x}^i_{[n_i]}) := n_i$.*
- *For each $n \in \mathbb{N}_+$ and all $\langle v_{n-1}, \ldots, v_1, v_0 \rangle \in \mathbb{B}_{[n]}$, let $v_{n-1} \ldots v_1 v_0 \in \mathcal{T}(V)$ and $\mathsf{width}(v_{n-1} \ldots v_1 v_0) := n$.*
- *If $s, t \in \mathcal{T}(V)$, then let $s \otimes t \in \mathcal{T}(V)$ and $\mathsf{width}(s \otimes t) := \mathsf{width}(s) + \mathsf{width}(t)$.*
- *If $t \in \mathcal{T}(V)$ and $i, j \in \mathbb{N}$ with $0 \leq i \leq j < \mathsf{width}(t)$, then let $t[j, i] \in \mathcal{T}(V)$ and $\mathsf{width}(t[j, i]) := j - i + 1$.*
- *If $t \in \mathcal{T}(V)$, then let $\mathsf{neg}(t) \in \mathcal{T}(V)$ and $\mathsf{width}(\mathsf{neg}(t)) := \mathsf{width}(t)$.*
- *If $a, b, s, t \in \mathcal{T}(V)$ with $\mathsf{width}(a) = \mathsf{width}(b)$ and $\mathsf{width}(s) = \mathsf{width}(t)$, then let*
  - *$\mathsf{ite}(a = b, s, t) \in \mathcal{T}(V)$ and $\mathsf{width}(\mathsf{ite}(a = b, s, t)) := \mathsf{width}(s)$,*
  - *$\mathsf{ite}(a < b, s, t) \in \mathcal{T}(V)$ and $\mathsf{width}(\mathsf{ite}(a < b, s, t)) := \mathsf{width}(s)$.*
- *For $s, t \in \mathcal{T}(V)$ with $\mathsf{width}(s) = \mathsf{width}(t)$ and $\odot \in \{\mathsf{and}, \mathsf{or}, \mathsf{xor}, \mathsf{nand}, \mathsf{nor}, \mathsf{xnor}\}$ let $s \odot t \in \mathcal{T}(V)$ and $\mathsf{width}(s \odot t) := \mathsf{width}(s)$.*
- *For $s, t \in \mathcal{T}(V)$ with $\mathsf{width}(s) = \mathsf{width}(t)$ and $\circledcirc \in \{\oplus, \ominus, \circledast\}$ let $s \circledcirc t \in \mathcal{T}(V)$ and $\mathsf{width}(s \circledcirc t) := \mathsf{width}(s)$.*
- *For $s, t \in \mathcal{T}(V)$ and $m \in \mathbb{N}_+$ with $\mathsf{width}(s) = 2^{\mathsf{width}(t)} \cdot m$ let $\mathsf{read}(s, t) \in \mathcal{T}(V)$ and $\mathsf{width}(\mathsf{read}(s, t)) := m$.*
- *For $s, t, u \in \mathcal{T}(V)$ with $\mathsf{width}(s) = 2^{\mathsf{width}(t)} \cdot \mathsf{width}(u)$ let $\mathsf{write}(s, t, u) \in \mathcal{T}(V)$ and $\mathsf{width}(\mathsf{write}(s, t, u)) := \mathsf{width}(s)$.*
- *If $t \in \mathcal{T}(V)$, then let $(t) \in \mathcal{T}(V)$ and $\mathsf{width}(\,(t)\,) := \mathsf{width}(t)$.* ■

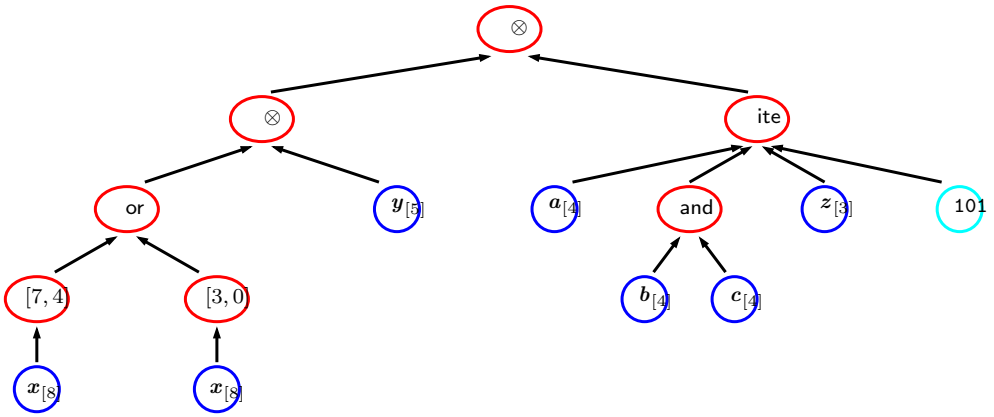
Example 5.2 (Bitvector Terms) *Let $V := \{\boldsymbol{a}_{[4]}, \boldsymbol{b}_{[4]}, \boldsymbol{c}_{[4]}, \boldsymbol{x}_{[8]}, \boldsymbol{y}_{[5]}, \boldsymbol{z}_{[3]}\}$ be a set of bitvector variables. Let t be the following term:*

$$\boldsymbol{x}_{[8]}[7,4] \text{ or } \boldsymbol{x}_{[8]}[3,0] \;\otimes\; \boldsymbol{y}_{[5]} \;\otimes\; \mathsf{ite}(\,\boldsymbol{a}_{[4]} = \boldsymbol{b}_{[4]} \text{ and } \boldsymbol{c}_{[4]},\, \boldsymbol{z}_{[3]},\, 101\,).$$

Then $t \in \mathcal{T}(V)$ and $\mathsf{width}(t) = 12$. □

Each bitvector term $t \in \mathcal{T}(V)$ can be represented as a finite directed tree reflecting the structure of sub-terms as determined by the inductive definition of bitvector terms. For each $t \in \mathcal{T}(V)$ this graph is called the *parse tree* of t.

The parse tree of the bitvector term occurring in Example 5.2 is shown in Figure 5.1 below. Leaf nodes are labeled with bitvector variables and bitvector constants, and the remaining nodes are labeled with bitvector operators. The edges between the nodes indicate the operator-argument relation defined by operators and sub-terms of t. Parentheses are not explicitly represented in the parse tree. The root node is called the *top-level operator* of t.



$$\big(\,(\,\boldsymbol{x}_{[8]}[7,4]\ \mathsf{or}\ \boldsymbol{x}_{[8]}[3,0]\,)\ \otimes\ \boldsymbol{y}_{[5]}\,\big)\ \otimes\ \mathsf{ite}\big(\,\boldsymbol{a}_{[4]} = \boldsymbol{b}_{[4]}\ \mathsf{and}\ \boldsymbol{c}_{[4]},\ \boldsymbol{z}_{[3]},\ 101\,\big)$$

Figure 5.1: Parse Tree Representation of Bitvector Terms

Parse tree representation of bitvector terms is not necessarily unique. Consider the following examples:

a) $\boldsymbol{v}_{[8]} \otimes \boldsymbol{x}_{[8]} \otimes \boldsymbol{y}_{[8]} \otimes \boldsymbol{z}_{[8]}$

b) $\boldsymbol{x}_{[8]}\ \mathsf{and}\ \boldsymbol{y}_{[8]}\ \mathsf{or}\ \boldsymbol{z}_{[8]}$

c) $\boldsymbol{x}_{[8]} \oplus \boldsymbol{y}_{[8]}\ \mathsf{xor}\ \boldsymbol{z}_{[8]}$

d) $\boldsymbol{v}_{[8]} \otimes \boldsymbol{x}_{[8]}\ \mathsf{and}\ \boldsymbol{y}_{[8]} \otimes \boldsymbol{z}_{[8]}$

e) $\boldsymbol{x}_{[8]} \otimes \boldsymbol{y}_{[8]}\ [3,0]$

In all five examples it is not obvious from the context which of the operators is the top-level operator, whereas for

f) $\boldsymbol{x}_{[4]} \otimes \boldsymbol{y}_{[4]}\ \mathsf{and}\ \boldsymbol{z}_{[8]}$

it can be concluded that and is the top-level operator because otherwise the term would not be a well-formed bitvector term due to the widths of the bitvector variables.

To avoid such ambiguities, we agree upon the following operator precedence for notation of bitvector terms.

Definition 5.3 (Operator Precedence and Use of Parentheses) *Let V be a finite set of bitvector variables. For textual representation of well-formed bitvector terms $t \in \mathcal{T}(V)$ we define the following operator precedence:*

- *extraction binds strongest*
- *bitwise Boolean connectives bind stronger than concatenation*
- *sequential concatenation binds from left to right*

We require that sub-terms of t must be parenthesized whenever otherwise the parse tree of t could not be uniquely determined from the textual representation of t. ■

We use $\equiv$ to denote that two bitvector terms are syntactically equal modulo parse tree representation. For example, reconsidering the previous sample bitvector terms, we then have:

a) $\boldsymbol{v}_{[8]} \otimes \boldsymbol{x}_{[8]} \otimes \boldsymbol{y}_{[8]} \otimes \boldsymbol{z}_{[8]} \quad \equiv \quad \big(\big(\boldsymbol{v}_{[8]} \otimes \boldsymbol{x}_{[8]} \big) \otimes \boldsymbol{y}_{[8]} \big) \otimes \boldsymbol{z}_{[8]}$

b) is ambiguous and must either be written as

$(\boldsymbol{x}_{[8]}$ and $\boldsymbol{y}_{[8]})$ or $\boldsymbol{z}_{[8]}$

or as

$\boldsymbol{x}_{[8]}$ and $(\boldsymbol{y}_{[8]}$ or $\boldsymbol{z}_{[8]})$

c) is ambiguous and must either be written as

$(\boldsymbol{x}_{[8]} \oplus \boldsymbol{y}_{[8]})$ xor $\boldsymbol{z}_{[8]}$

or as

$\boldsymbol{x}_{[8]} \oplus (\boldsymbol{y}_{[8]}$ xor $\boldsymbol{z}_{[8]})$

d) $\boldsymbol{v}_{[8]} \otimes \boldsymbol{x}_{[8]}$ and $\boldsymbol{y}_{[8]} \otimes \boldsymbol{z}_{[8]} \quad \equiv \quad \boldsymbol{v}_{[8]} \otimes (\, \boldsymbol{x}_{[8]}$ and $\boldsymbol{y}_{[8]}\,) \otimes \boldsymbol{z}_{[8]}$

and

$\boldsymbol{v}_{[8]} \otimes \boldsymbol{x}_{[8]}$ and $\boldsymbol{y}_{[8]} \otimes \boldsymbol{z}_{[8]} \quad \not\equiv \quad (\, \boldsymbol{v}_{[8]} \otimes \boldsymbol{x}_{[8]}\,)$ and $(\, \boldsymbol{y}_{[8]} \otimes \boldsymbol{z}_{[8]}\,)$

e) $\boldsymbol{x}_{[8]} \otimes \boldsymbol{y}_{[8]}\,[3,0] \quad \equiv \quad \boldsymbol{x}_{[8]} \otimes (\, \boldsymbol{y}_{[8]}[3,0]\,)$ with width 12

and

$\boldsymbol{x}_{[8]} \otimes \boldsymbol{y}_{[8]}\,[3,0] \quad \not\equiv \quad (\, \boldsymbol{x}_{[8]} \otimes \boldsymbol{y}_{[8]}\,)[3,0]$ with width 4.

We furthermore define two characteristic properties which provide a means to measure the complexity of bitvector terms. The *depth* of a bitvector term indicates the maximum nesting of operators.

Definition 5.4 (Depth of a Bitvector Term) *Let V be a finite set of bitvector variables. The depth $\mathsf{depth}(t)$ of a bitvector term $t \in \mathcal{T}(V)$ is defined inductively.*

- *For each $\boldsymbol{x}_{[n]} \in V$:*
  - $\mathsf{depth}(\boldsymbol{x}_{[n]}) := 0$.
- *For each $\langle v_{n-1}, \ldots, v_0 \rangle \in \mathbb{B}_{[*]}$:*
  - $\mathsf{depth}(v_{n-1} \ldots v_0) := 0$.
- *For $s, t \in \mathcal{T}(V)$:*
  - $\mathsf{depth}(s \otimes t) := \max\{\mathsf{depth}(s), \mathsf{depth}(t)\} + 1$.
- *For $t \in \mathcal{T}(V)$ and $i, j \in \mathbb{N}$ with $0 \leq i \leq j < \mathsf{depth}(t)$:*
  - $\mathsf{depth}(t[j, i]) := \mathsf{depth}(t) + 1$.
- *For $t \in \mathcal{T}(V)$:*
  - $\mathsf{depth}(\mathsf{neg}(t)) := \mathsf{depth}(t) + 1$.
- *For $a, b, s, t \in \mathcal{T}(V)$ with $\mathsf{width}(a) = \mathsf{width}(b)$ and $\mathsf{width}(s) = \mathsf{width}(t)$:*
  - $\mathsf{depth}(\mathsf{ite}(a = b, s, t)) := \max\{\mathsf{depth}(a), \mathsf{depth}(b), \mathsf{depth}(s), \mathsf{depth}(t)\} + 1$.
  - $\mathsf{depth}(\mathsf{ite}(a < b, s, t)) := \max\{\mathsf{depth}(a), \mathsf{depth}(b), \mathsf{depth}(s), \mathsf{depth}(t)\} + 1$.
- *For $s, t \in \mathcal{T}(V)$ with $\mathsf{width}(s) = \mathsf{width}(t)$ and $\odot \in \{\mathsf{and}, \mathsf{or}, \mathsf{xor}, \mathsf{nand}, \mathsf{nor}, \mathsf{xnor}\}$:*
  - $\mathsf{depth}(s \odot t) := \max\{\mathsf{depth}(s), \mathsf{depth}(t)\} + 1$.
- *For $s, t \in \mathcal{T}(V)$ with $\mathsf{width}(s) = \mathsf{width}(t)$ and $\circledcirc \in \{\oplus, \ominus, \circledast\}$:*
  - $\mathsf{depth}(s \circledcirc t) := \max\{\mathsf{depth}(s), \mathsf{depth}(t)\} + 1$.
- *For $s, t \in \mathcal{T}(V)$ and $m \in \mathbb{N}_+$ with $\mathsf{width}(s) = 2^{\mathsf{width}(t)} \cdot m$:*
  - $\mathsf{depth}(\mathsf{read}(s, t)) := \max\{\mathsf{depth}(s), \mathsf{depth}(t)\} + 1$.
- *For $s, t, u \in \mathcal{T}(V)$ with $\mathsf{width}(s) = 2^{\mathsf{width}(t)} \cdot \mathsf{width}(u)$:*
  - $\mathsf{depth}(\mathsf{write}(s, t, u)) := \max\{\mathsf{depth}(s), \mathsf{depth}(t), \mathsf{depth}(u)\} + 1$.
- *If $t \in \mathcal{T}(V)$, then let $\mathsf{depth}(\,(t)\,) := \mathsf{depth}(t)$.* ■

The value of $\mathsf{depth}(t)$ for a bitvector term $t \in \mathcal{T}(V)$ yields the depth of the corresponding parse tree of t.

As a next step, the *length of* a bitvector term is defined, indicating the overall number of operators occurring in the term, i.e. the sum of the number of bitvector variables, bitvector constants and bitvector operators occurring in t.

Definition 5.5 (Length of a Bitvector Term) *Let V be a finite set of bitvector variables. The length* $\mathsf{length}(t)$ *of a bitvector term $t \in \mathcal{T}(V)$ is defined inductively.*

- *For each $\boldsymbol{x}_{[n]} \in V$:*
  - $\mathsf{length}(\boldsymbol{x}_{[n]}) := 1$.
- *For each $\langle v_{n-1}, \ldots, v_0 \rangle \in \mathbb{B}_{[*]}$:*
  - $\mathsf{length}(v_{n-1} \ldots v_0) := 1$.
- *For $s, t \in \mathcal{T}(V)$:*
  - $\mathsf{length}(s \otimes t) := \mathsf{length}(s) + \mathsf{length}(t) + 1$.
- *For $t \in \mathcal{T}(V)$ and $i, j \in \mathbb{N}$ with $0 \leq i \leq j < \mathsf{length}(t)$:*
  - $\mathsf{length}(t[j, i]) := \mathsf{length}(t) + 1$.
- *For $t \in \mathcal{T}(V)$:*
  - $\mathsf{length}(\mathsf{neg}(t)) := \mathsf{length}(t) + 1$.
- *For $a, b, s, t \in \mathcal{T}(V)$ with $\mathsf{width}(a) = \mathsf{width}(b)$ and $\mathsf{width}(s) = \mathsf{width}(t)$:*
  - $\mathsf{length}(\mathsf{ite}(a = b, s, t)) := \mathsf{length}(a) + \mathsf{length}(b) + \mathsf{length}(s) + \mathsf{length}(t) + 1$.
- *For $s, t \in \mathcal{T}(V)$ with $\mathsf{width}(s) = \mathsf{width}(t)$ and $\odot \in \{\mathsf{and}, \mathsf{or}, \mathsf{xor}, \mathsf{nand}, \mathsf{nor}, \mathsf{xnor}\}$:*
  - $\mathsf{length}(s \odot t) := \mathsf{length}(s) + \mathsf{length}(t) + 1$.
- *For $s, t \in \mathcal{T}(V)$ with $\mathsf{width}(s) = \mathsf{width}(t)$ and $\circledcirc \in \{\oplus, \ominus, \circledast\}$:*
  - $\mathsf{length}(s \circledcirc t) := \mathsf{length}(s) + \mathsf{length}(t) + 1$.
- *For $s, t \in \mathcal{T}(V)$ and $m \in \mathbb{N}_+$ with $\mathsf{width}(s) = 2^{\mathsf{width}(t)} \cdot m$:*
  - $\mathsf{length}(\mathsf{read}(s, t)) := \mathsf{length}(s) + \mathsf{length}(t) + 1$.
- *For $s, t, u \in \mathcal{T}(V)$ with $\mathsf{width}(s) = 2^{\mathsf{width}(t)} \cdot \mathsf{width}(u)$:*
  - $\mathsf{length}(\mathsf{write}(s, t, u)) := \mathsf{length}(s) + \mathsf{length}(t) + \mathsf{length}(u) + 1$.
- *If $t \in \mathcal{T}(V)$, then let* $\mathsf{length}(\,(t)\,) := \mathsf{length}(t)$. ■

The value of $\mathsf{length}(t)$ for a bitvector term $t \in \mathcal{T}(V)$ yields the overall number of nodes in the parse tree of t.

Example 5.6 (Length and Depth of Bitvector Terms) *Let t be the bitvector term defined in Example 5.2 and illustrated in Figure 5.1. Then* $\mathsf{length}(t) = 15$ *and* $\mathsf{depth}(t) = 4$. □

5.2 Interpretation of Bitvector Terms by Bitvector Functions

Bitvector terms are interpreted by bitvector functions. Semantics of a term $t \in \mathcal{T}(V)$ is given by the bitvector function of width $\mathsf{width}(t)$ which is defined inductively in the following way:

Definition 5.7 (Interpretation of Bitvector Terms) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables of widths $n_1, \ldots, n_k$. Let $t \in \mathcal{T}(V)$ be a bitvector term and let $n \in \mathbb{N}_+$ with $n := \mathsf{width}(t)$. The **interpretation** $[\![\, t \,]\!]$ of t is the k-ary bitvector function*

$$[\![\, t \,]\!] : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$$

of width n on bitvectors of widths $n_1, \ldots, n_k$ which is defined by induction on the term structure of t in the following way:

- *if $t \equiv \boldsymbol{x}^i_{[n_i]}$, then $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) := \boldsymbol{x}^i_{[n_i]}$*
- *if $t \equiv v_{n-1} \ldots v_1 v_0$, then $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) := \langle v_{n-1}, \ldots, v_1, v_0 \rangle$*
- *if $t \equiv t_1 \otimes t_2$ with $t_1, t_2 \in \mathcal{T}(V)$, then*
  $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) := [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \otimes [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$
- *if $t \equiv s[j,i]$ with $s \in \mathcal{T}(V)$ and $i, j \in \mathbb{N}_+$, then*
  $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) := (\, [\![\, s \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \,)[j,i]$
- *if $t \equiv \mathsf{neg}(s)$ with $s \in \mathcal{T}(V)$, then*
  $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) := \mathsf{neg}(\, [\![\, s \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \,)$
- *if $t \equiv t_1 \odot t_2$ with $t_1, t_2 \in \mathcal{T}(V)$ and $\odot \in \{\mathsf{and}, \mathsf{or}, \mathsf{xor}, \mathsf{nand}, \mathsf{nor}, \mathsf{xnor}\}$, then*
  $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) := [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \odot [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$
- *if $t \equiv \mathsf{ite}(t_1 = t_2, t_3, t_4)$ with $t_1, t_2, t_3, t_4 \in \mathcal{T}(V)$, then*
  $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) := \mathsf{ite}(\, [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}),$
  $[\![\, t_3 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}), [\![\, t_4 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \,)$
- *if $t \equiv \mathsf{ite}(t_1 < t_2, t_3, t_4)$ with $t_1, t_2, t_3, t_4 \in \mathcal{T}(V)$, then*
  $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) := \mathsf{ite}(\, [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) < [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}),$
  $[\![\, t_3 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}), [\![\, t_4 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \,)$
- *if $t \equiv t_1 \circledcirc t_2$ with $t_1, t_2 \in \mathcal{T}(V)$ and $\circledcirc \in \{\oplus, \ominus, \circledast\}$, then*
  $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) := [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \circledcirc [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$
- *if $t \equiv \mathsf{read}(t_1, t_2)$ with $t_1, t_2 \in \mathcal{T}(V)$, then $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) :=$*
  $\mathsf{read}([\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}), [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}))$
- *if $t \equiv \mathsf{write}(t_1, t_2, t_3)$ with $t_1, t_2, t_3 \in \mathcal{T}(V)$, then $[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) :=$*
  $\mathsf{write}([\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}), [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}), [\![\, t_3 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}))$

for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$. ■

If $\mathcal{F}_{[n]}$ is a bitvector function and $t \in \mathcal{T}(V)$ is a bitvector term such that $\mathcal{F}_{[n]} = [\![t]\!]$ then we say that $\mathcal{F}_{[n]}$ is the *interpretation* of t and that t *represents* $\mathcal{F}_{[n]}$.

Note: The representation relation between bitvector terms and bitvector functions is not one-to-one. Every bitvector term $t \in \mathcal{T}(V)$ has a unique interpretation as a bitvector function with respect to V, but in general there exist numerous terms that represent the same bitvector function.

In order to determine the bitvector value which is represented by a bitvector term for given specific values of all bitvector variables occurring in the term, we define the notion of *valuations* of a set of bitvector variables V.

Definition 5.8 (Valuations of Bitvector Variables) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of k bitvector variables of widths $n_1, \ldots, n_k$. Then each k-tuple $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle$ with $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ is a **valuation** of V.* ■

Definition 5.9 (Evaluation of Bitvector Terms) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of k bitvector variables of widths $n_1, \ldots, n_k$. Let $t \in \mathcal{T}(V)$ be a bitvector term over variables of V. Let $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$. Then the **evaluation** of t for $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle$ is given by*

$$\boldsymbol{y}_{[m]} \;:=\; [\![t]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$$

where $m = \mathsf{width}(t)$. ■

5.3 Systems of Bitvector Equations

In this section, the notion of *bitvector equations* and *systems of bitvector equations* is introduced. Within our framework, comparison of two bitvectors is restricted to comparison of bitvectors of the same width. An equation of two bitvector terms is defined a follows:

Definition 5.10 (Bitvector Equations) *Let V be a finite set of bitvector variables. A **bitvector equation** is a pair*

$$e \;=\; \langle t_1, t_2 \rangle$$

of bitvector terms $t_1, t_2 \in \mathcal{T}(V)$ with $\mathsf{width}(t_1) = \mathsf{width}(t_2)$. ■

For a given finite set V of bitvector variables, let

$$\mathcal{E}(V) \;:=\; \{\, \langle t_1, t_2 \rangle \;\mid\; t_1, t_2 \in \mathcal{T}(V) \;\wedge\; \mathsf{width}(t_1) = \mathsf{width}(t_2) \,\}$$

denote the set of all bitvector equations of bitvector terms over V. As a textual notation of bitvector equations we will normally rather write $t_1 = t_2$, especially if t_1 and t_2 are explicitly stated, instead of writing $\langle t_1, t_2 \rangle$.

Example 5.11 (Bitvector Equations) *Let* $V := \{ \boldsymbol{a}_{[4]}, \boldsymbol{b}_{[4]}, \boldsymbol{x}_{[8]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[8]} \}$. *Then*

$$\boldsymbol{x}_{[8]} \text{ and } \boldsymbol{y}_{[8]} \quad = \quad \boldsymbol{a}_{[4]} \otimes \boldsymbol{b}_{[4]}$$

is a bitvector equation of $\mathcal{E}(V)$. □

A *system of bitvector equations* is a finite collection of equations over the same set of bitvector variables.

Definition 5.12 (Systems of Bitvector Equations) *Let* V *be a finite set of bitvector variables. A* ***system of bitvector equations*** *is a finite set*

$$E \subseteq \mathcal{E}(V)$$

of bitvector equations over V. ■

To provide a measurement for the complexity of bitvector equations and of systems of equations, we define the *length* of a bitvector equation as the sum of the lengths of the left hand side term and the right hand side term.

Definition 5.13 (Length of a Bitvector Equation) *Let* V *be a finite set of bitvector variables. The* ***length*** $\mathsf{length}(e)$ *of a bitvector equation* $e \in \mathcal{E}(V)$ *with* $e = \langle t_1, t_2 \rangle$ *for bitvector terms* $t_1 \in \mathcal{T}(V)$ *and* $t_2 \in \mathcal{T}(V)$ *is defined as*

$$\mathsf{length}(e) \;:=\; \mathsf{length}(t_1) + \mathsf{length}(t_2)$$

■

The length of a system of bitvector equations then is the sum of the lengths of all equations.

Definition 5.14 (Length of a System of Bitvector Equations) *Let* V *be a finite set of bitvector variables. The* ***length*** $\mathsf{length}(E)$ *of a system of bitvector equations* $E \subseteq \mathcal{E}(V)$ *is defined as*

$$\mathsf{length}(E) \;:=\; \sum_{e \in E} \mathsf{length}(e)$$

■

5.4 Satisfiability of Bitvector Equations

A bitvector equation $e = \langle t_1, t_2 \rangle$ is said to be *satisfiable* if there exists a valuation of all bitvector variables such that both terms t_1 and t_2 evaluate to the same bitvector.

Definition 5.15 (Satisfiability of Bitvector Equations) *Let $n_1, \ldots, n_k \in \mathbb{N}_+$ with $k \in \mathbb{N}_+$ and let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables. Let $e = \langle t_1, t_2 \rangle \in \mathcal{E}(V)$ be a bitvector equation. Then we define:*

$$e \text{ is satisfiable} \quad :\Longleftrightarrow$$
$$\exists\ \boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]} \ :\ [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \quad \blacksquare$$

If $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ are bitvectors for which the interpreting functions $[\![\, t_1 \,]\!]$ and $[\![\, t_2 \,]\!]$ yield the same value, then the k-tuple $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle$ is called a *satisfying solution* of the bitvector equation $t_1{=}t_2$.

Definition 5.16 (Satisfying Solutions of Bitvector Equations) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of k bitvector variables and let $e = \langle t_1, t_2 \rangle \in \mathcal{E}(V)$ be a bitvector equation. The set $\mathcal{L}(e)$ of satisfying solutions of e is defined as:*

$$\mathcal{L}(e) \ :=\ \{\, \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \ \mid\ [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \,\}$$

We say that e is satisfiable if $\mathcal{L}(e) \neq \emptyset$, and e is valid if $\mathcal{L}(e) = \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]}$. $\blacksquare$

Example 5.17 (Bitvector Equations – Satisfiability) *Let $\boldsymbol{x}_{[4]}$ and $\boldsymbol{y}_{[4]}$ be bitvector variables and consider the following bitvector equation of $\mathcal{E}(\boldsymbol{x}_{[4]}, \boldsymbol{y}_{[4]})$:*

$$\boldsymbol{x}_{[4]} \ \mathsf{and}\ \mathtt{1100} \ =\ \boldsymbol{y}_{[4]} \tag{5.1}$$

Equation (5.1) is satisfiable, e.g. by $\boldsymbol{x}_{[4]} := \mathtt{0111}$ and $\boldsymbol{y}_{[4]} := \mathtt{0100}$, but not valid, as $\boldsymbol{x}_{[4]} := \mathtt{1111}$ and $\boldsymbol{y}_{[4]} := \mathtt{0000}$ is not a satisfying solution. $\square$

Example 5.18 (Bitvector Equations – Unsatisfiability) *Let $\boldsymbol{x}_{[8]}$ be a bitvector variable and consider the bitvector equation:*

$$\boldsymbol{x}_{[8]} \ =\ \mathsf{neg}(\boldsymbol{x}_{[8]}) \tag{5.2}$$

Equation (5.2) is unsatisfiable. $\square$

Example 5.19 (Bitvector Equations – Validity) *Let $\boldsymbol{x}_{[16]}$ be a bitvector variable and consider the bitvector equation:*

$$\boldsymbol{x}_{[16]}[15, 8] \otimes \boldsymbol{x}_{[16]}[7, 0] \ =\ \boldsymbol{x}_{[16]} \tag{5.3}$$

As easily can be see, Equation (5.3) is a tautology, i.e. is universally valid. $\square$

We now define satisfiability for systems of bitvector equations. A system E of bitvector equations is satisfiable if there exists a valuation of the bitvector variables such that all equations of E hold simultaneously. Again, validity is defined correspondingly.

Definition 5.20 (Satisfiability of Systems of Bitvector Equations) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$ and let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables. Let $E \subseteq \mathcal{E}(V)$ be a system of bitvector equations. The set $\mathcal{L}(E)$ of satisfying solutions of E is defined as*

$$\mathcal{L}(E) := \{\, \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \mid \forall\, e \in E \;:\; \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(e) \,\}$$

E is satisfiable if $\mathcal{L}(E) \neq \emptyset$, and E is valid if $\mathcal{L}(E) = \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]}$. ■

Example 5.21 (Systems of Bitvector Equations) *Let $\boldsymbol{x}_{[8]}$ and $\boldsymbol{y}_{[4]}$ be bitvector variables. Consider the following system of bitvector equations:*

$$\begin{array}{lcl} \boldsymbol{x}_{[8]} & = & \boldsymbol{y}_{[4]} \otimes \boldsymbol{y}_{[4]} \\ \boldsymbol{x}_{[8]}[7,4] & = & \mathsf{neg}(\boldsymbol{x}_{[8]}[3,0]) \end{array} \tag{5.4}$$

Taken separately, the first equation as well as the second equation is satisfiable. However, system (5.4) as a whole is unsatisfiable because the second equation requires $\boldsymbol{x}_{[8]}[7,4]$ and $\boldsymbol{x}_{[8]}[3,0]$ to have complementary values, whereas the first equation demands that both values must be the same. □

Sets of satisfying solutions of bitvector equations can exactly be characterized by *zeros* of bitvector functions.

Definition 5.22 (Zeros of Bitvector Functions) *Let $n, k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ be a bitvector function. Let $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ such that*

$$\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;=\; 0_{[n]}$$

*Then $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle$ is called a **zero** of $\mathcal{F}_{[n]}$.* ■

For each bitvector equation e there exists a bitvector function $\mathcal{F}_{[n]}$ such that $\mathcal{L}(e)$ is exactly the set of zeros of $\mathcal{F}_{[n]}$.

Corollary 5.23 (Satisfiability of Bitvector Equations) *Let $n_1, \ldots, n_k \in \mathbb{N}_+$ for a $k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of k bitvector variables. Let $e = \langle t_1, t_2 \rangle \in \mathcal{E}(V)$ be a bitvector equation. Let $n := \mathsf{width}(t_1)$. Then $n = \mathsf{width}(t_2)$ and there exists a k-ary bitvector function $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ such that*

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(e) \iff \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[n]}$$

holds for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$. ■

Proof: Let $e \in \mathcal{E}(V)$ with $e = \langle t_1, t_2 \rangle$ for $t_1, t_2 \in \mathcal{T}(V)$. Let $n := \mathsf{width}(t_1)$ and define $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ by

$$\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;:=\; [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;\mathsf{xor}\; [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$$

for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$. Then we have:

$$\begin{aligned}
\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;=\; 0_{[n]} \quad &\Longleftrightarrow \quad [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;\mathsf{xor}\; [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;=\; 0_{[n]} \\
&\Longleftrightarrow \quad [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;=\; [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \\
&\Longleftrightarrow \quad \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(e)
\end{aligned}$$

■

The same statement holds for systems of bitvector equations. The set of satisfying solutions of a system of bitvector equations can be characterized as the set of zeros of a specific bitvector function.

Corollary 5.24 (Satisfiability of Systems of Bitvector Equations) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables. Let $E \subseteq \mathcal{E}(V)$ be a system of bitvector equations. Then there exists $n \in \mathbb{N}_+$ and a k-ary bitvector function $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ such that*

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(E) \quad \Longleftrightarrow \quad \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[n]}$$

for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$. ■

Proof: Let $E = \{e_1, \ldots, e_p\}$ for $p := |E|$ and $e_1, \ldots, e_p \in \mathcal{E}(V)$. For each $i \in \{1, \ldots, p\}$ let $m_i := \mathsf{width}(e_i)$ and let $\mathcal{F}^i_{[m_i]}$ be defined according to Corollary 5.23. Let

$$n := \sum_{i=1}^{p} m_i$$

and define $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ by

$$\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;:=\; \mathcal{F}^1_{[m_1]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;\otimes\; \ldots \;\otimes\; \mathcal{F}^p_{[m_p]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$$

for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$. Then we have:

$$\begin{aligned}
\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[n]} \quad &\Longleftrightarrow \quad \forall\, i \in \{1, \ldots, p\} \;:\; \mathcal{F}^i_{[m_i]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[m_i]} \\
&\Longleftrightarrow \quad \forall\, i \in \{1, \ldots, p\} \;:\; \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(e_i) \\
&\Longleftrightarrow \quad \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(E)
\end{aligned}$$

■

The proofs of Corollaries 5.23 and 5.24 yield that for each system E of bitvector equations there exists a single bitvector equation with the identical set of satisfying solutions as E.

Corollary 5.25 (Satisfiability of Systems of Bitvector Equations) *Let V be a finite set of bitvector variables. Let $E \subseteq \mathcal{E}(V)$ be a system of bitvector equations. Then there exists a bitvector equation $e \in \mathcal{E}(V)$ such that $\mathcal{L}(E) \;=\; \mathcal{L}(e)$.* ■

Proof: Let $E = \{e_1, \ldots, e_p\}$ for $p := |E|$ and $e_1, \ldots, e_p \in \mathcal{E}(V)$. Let $s_1, \ldots, s_p \in \mathcal{T}(V)$ and $t_1, \ldots, t_p \in \mathcal{T}(V)$ such that $e_i = \langle s_i, t_i \rangle$ for all $i \in \{1, \ldots, p\}$. Furthermore let $m_i := \mathsf{width}(e_i)$ for each $i \in \{1, \ldots, p\}$ and let

$$n := \sum_{i=1}^{p} \mathsf{width}(t_i)$$

Then define

$$t \;:\equiv\; (s_1 \;\mathsf{xor}\; t_1) \;\otimes\; \ldots \;\otimes\; (s_p \;\mathsf{xor}\; t_p)$$

Then $t \in \mathcal{T}(V)$, $\mathsf{width}(t) = n$, and for $e := \langle t, 0_{[n]} \rangle$ we have $\mathcal{L}(E) \;=\; \mathcal{L}(e)$. ■

5.5 Checking Satisfiability of Systems of Bitvector Equations

Corollary 5.24 states that for each system E of bitvector equations there exists a bitvector function $\mathcal{F}_{[n]}$ such that satisfying solutions of E are exactly the zeros of $\mathcal{F}_{[n]}$, i.e.

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(E) \quad \Longleftrightarrow \quad \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[n]} \tag{5.5}$$

Systems of bitvector equations are used in RTL model checking to describe functionality of digital circuits. For a given circuit design D and a bounded circuit property P, a system E of bitvector equations can be synthesized such that P holds for D if and only if E is unsatisfiable and such that each satisfying solution of E yields a counterexample violating P for D.

Given such a system of bitvector equations E, a bitvector function $\mathcal{F}_{[n]}$ for which (5.5) holds can be constructed by syntactical analysis of the equations of E. The common approach in Bounded Model Checking is to represent $\mathcal{F}_{[n]}$ as a Boolean formula φ involving Boolean variables for each bit of each bitvector variable occurring in E. Existence of zeros for $\mathcal{F}_{[n]}$ is then determined by checking satisfiability of φ, usually done by employing SAT solvers and BDD techniques. However, the complexity of the best currently known algorithms for checking satisfiability of arbitrary Boolean formulae is at least super-polynomial in the number of Boolean variables occurring in the formulae.

Assume that all bitvector variables occurring in E have the same width, and assume that the bitvector function $\mathcal{F}_{[n]}$ occurring in (5.5) is a bitwise bitvector function. Then satisfiability of E corresponds to satisfiability of the bitwise problem **BvSAT**$(\mathcal{F}_{[n]})$, and the reduction technique presented in Chapter 4 can be applied to reduce the width of the bitvector variables of E and to reduce the number of Boolean variables occurring in φ. This can greatly influence verification runtimes. The results about reductions of bitwise **BvSAT** problems then guarantee a one-to-one correspondence of verification results and furthermore allow for an easy handling of counterexamples. Unfortunately, usually $\mathcal{F}_{[n]}$ is not bitwise. In the following chapters we

show that yet each bitvector function can always be decomposed into bitwise parts. If – as it is the case in our framework – $\mathcal{F}_{[n]}$ is (implicitly) given by a system of bitvector equations E, then such a decomposition can be computed by syntactical analysis of the terms of E. In Chapter 6, a decomposition technique and the appropriate procedures are presented which allow to deconstruct the general **BvSAT** problem associated with (5.5) into a collection of bitwise **BvSAT** problems for which the proposed reduction technique can be applied separately.

Chapter 6

Bitwise Decompositions of Systems of Bitvector Equations

In the last chapter we have seen how satisfying solutions of systems of bitvector equations can be characterized as sets of zeros of bitvector functions. Hence, satisfiability of systems of bitvector equations can be reduced to existence of zeros of specific bitvector functions.

The **BvSAT** problem, which has been introduced in Chapter 4, asks whether or not a given bitvector function has a zero. For bitwise **BvSAT** problems and bitwise bitvector functions we have presented a width reduction technique which strictly preserves satisfiability and existence of zeros. As complexity of checking satisfiability of bitvector equations depends on the widths of the bitvector variables, the presented results can be used to simplify such satisfiability checks.

Unfortunately, the bitvector functions and **BvSAT** problems which characterize satisfiability of arbitrarily given systems of bitvector equations are not bitwise in general. However, bitwise data dependencies can be found between complete bitvector variables or can exist only between smaller parts of the variables or just between single bits. Often bitvector functions are at least partially bitwise on certain contiguous subranges of the bits of the bitvectors.

In this chapter, we show that a **BvSAT** problem, defined by an arbitrary bitvector function, can always be decomposed into an equivalent conjunction of independent bitwise **BvSAT** problems. Thus, the satisfiability problem related to a system of bitvector equations can always be decomposed into a collection of independent satisfiability problems which are all bitwise.

In the following sections of this chapter we introduce the formal concept of *partially bitwise bitvector functions* and *bitwise decompositions* of **BvSAT** problems. We present a way to compute a bitwise decomposition of an arbitrary bitvector function $\mathcal{F}_{[n]}$ from the set of partially bitwise portions of $\mathcal{F}_{[n]}$. Subsequently, in Chapter 7, we show that, if $\mathcal{F}_{[n]}$ is related to a system of bitvector equations E in the way that zeros of $\mathcal{F}_{[n]}$ are satisfying solutions of E, then the partially bitwise portions of $\mathcal{F}_{[n]}$ can be determined by syntactical analysis of the equations and bitvector terms of E. The basic concept of this approach is illustrated in the following motivating example.

6.1 Motivating Example

Let $V := \{\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}\}$ be a set of bitvector variables and assume that the following system $E \subseteq \mathcal{E}(V)$ of bitvector equations is given:

$$\begin{aligned} \boldsymbol{x}_{[16]}[7,2] &= \boldsymbol{a}_{[8]}[5,0] \text{ or } \boldsymbol{b}_{[8]}[5,0] \\ \boldsymbol{z}_{[12]}[7,0] &= \boldsymbol{x}_{[16]}[11,4] \text{ and } \boldsymbol{y}_{[8]} \end{aligned} \tag{6.1}$$

According to Corollary 5.24, there exists $n \in \mathbb{N}_+$ and a bitvector function

$$\mathcal{F}_{[n]} : \mathbb{B}_{[8]} \times \mathbb{B}_{[8]} \times \mathbb{B}_{[16]} \times \mathbb{B}_{[8]} \times \mathbb{B}_{[12]} \longrightarrow \mathbb{B}_{[n]}$$

such that for all $\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]} \in \mathbb{B}_{[8]}$, $\boldsymbol{x}_{[16]} \in \mathbb{B}_{[16]}$, $\boldsymbol{y}_{[8]} \in \mathbb{B}_{[8]}$ and $\boldsymbol{z}_{[12]} \in \mathbb{B}_{[12]}$ we have:

$$\langle \boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]} \rangle \;\; \textit{satisfies} \;\; (6.1) \quad \Longleftrightarrow \quad \mathcal{F}_{[n]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}) \;=\; 0_{[n]}$$

Satisfiability of (6.1) is thus described by satisfiability of the **BvSAT** problem for $\mathcal{F}_{[n]}$. In this case, $\mathcal{F}_{[n]}$ can be constructed as follows (cf. proof of Corollary 5.25): let $e_1 \in \mathcal{E}(V)$ be the first equation of E, that is e_1 is given by:

$$\boldsymbol{x}_{[16]}[7,2] = \boldsymbol{a}_{[8]}[5,0] \text{ or } \boldsymbol{b}_{[8]}[5,0] \tag{6.2}$$

The bit-level data flow and the bit-level data dependencies between the individual bits of $\boldsymbol{a}_{[8]}$, $\boldsymbol{b}_{[8]}$ and $\boldsymbol{x}_{[16]}$, as imposed by e_1, are illustrated in Figure 6.1 below (the notion of *bit-level data flow* and *bit-level data dependencies imposed by a bitvector equation* refers to the *cone of influence* of the respective individual bits of the considered bitvector variables, here indicated by arrows for $\boldsymbol{x}_{[16]}[2], \boldsymbol{x}_{[16]}[3], \ldots, \boldsymbol{x}_{[16]}[7]$).

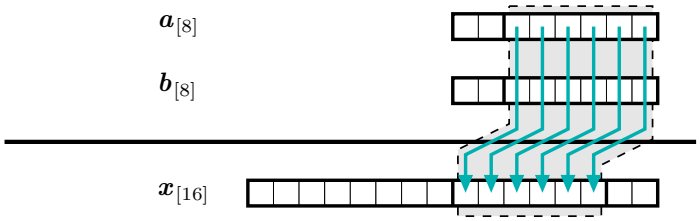


Figure 6.1: Data Flow of Equation (6.2)

For all $\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]} \in \mathbb{B}_{[8]}$ and $\boldsymbol{x}_{[16]} \in \mathbb{B}_{[16]}$, let $\mathcal{F}^1_{[6]} : \mathbb{B}_{[8]} \times \mathbb{B}_{[8]} \times \mathbb{B}_{[16]} \longrightarrow \mathbb{B}_{[6]}$ be defined by:

$$\mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}) \;:=\; (\, \boldsymbol{a}_{[8]}[5,0] \text{ or } \boldsymbol{b}_{[8]}[5,0] \,) \text{ xor } \boldsymbol{x}_{[16]}[7,2] \tag{6.3}$$

Then we have:

$$\langle \boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]} \rangle \;\; \textit{satisfies} \;\; (6.2) \qquad \Longleftrightarrow \qquad \mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}) \;=\; 0_{[6]}$$

Let $e_2 \in \mathcal{E}(V)$ be the second equation of E, i.e. let e_2 be given by:

$$\boldsymbol{z}_{[12]}[7,0] \;\;=\;\; \boldsymbol{x}_{[16]}[11,4] \text{ and } \boldsymbol{y}_{[8]} \tag{6.4}$$

The bit-level data flow of e_2 and the dependencies between the individual bits of $\boldsymbol{x}_{[16]}$, $\boldsymbol{y}_{[8]}$ and $\boldsymbol{z}_{[12]}$ are shown in Figure 6.2.

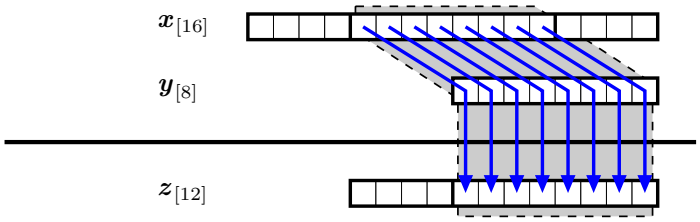


Figure 6.2: Data Flow of Equation (6.4)

Let $\mathcal{F}^2_{[8]} : \mathbb{B}_{[16]} \times \mathbb{B}_{[8]} \times \mathbb{B}_{[12]} \longrightarrow \mathbb{B}_{[8]}$ be defined by

$$\mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}) \;:=\; (\, \boldsymbol{x}_{[16]}[11,4] \text{ and } \boldsymbol{y}_{[8]} \,) \text{ xor } \boldsymbol{z}_{[12]}[7,0] \tag{6.5}$$

for all $\boldsymbol{x}_{[16]} \in \mathbb{B}_{[16]}$, $\boldsymbol{y}_{[8]} \in \mathbb{B}_{[8]}$ and $\boldsymbol{z}_{[12]} \in \mathbb{B}_{[12]}$. Then we have:

$$\langle \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]} \rangle \;\; \textit{satisfies} \;\; (6.4) \qquad \Longleftrightarrow \qquad \mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}) \;=\; 0_{[8]}$$

Now, let $\mathcal{F}_{[14]} : \mathbb{B}_{[8]} \times \mathbb{B}_{[8]} \times \mathbb{B}_{[16]} \times \mathbb{B}_{[8]} \times \mathbb{B}_{[12]} \longrightarrow \mathbb{B}_{[14]}$ be defined by:

$$\mathcal{F}_{[14]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}) \;\;:=\;\; \mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}) \;\otimes\; \mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}) \tag{6.6}$$

Then we have

$$\begin{aligned}
& \langle \boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]} \rangle \;\; \textit{satisfies} \;\; (6.1) \\
\Longleftrightarrow \quad & \langle \boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]} \rangle \;\; \textit{satisfies} \;\; (6.2) \;\; \textit{and} \;\; (6.4) \\
\Longleftrightarrow \quad & \mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}) \;=\; 0_{[6]} \;\; \wedge \;\; \mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}) \;=\; 0_{[8]} \\
\Longleftrightarrow \quad & \mathcal{F}_{[14]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}) \;=\; 0_{[14]}
\end{aligned}$$

and thus satisfying solutions of (6.1) are satisfying solutions of **BvSAT**$(\mathcal{F}_{[14]})$ and vice versa. However, $\mathcal{F}_{[14]}$ is not a bitwise bitvector function. The data dependencies of $\mathcal{F}_{[14]}$ are shown in Figure 6.3 which subsumes the data flow dependencies shown in Figures 6.1 and 6.2.

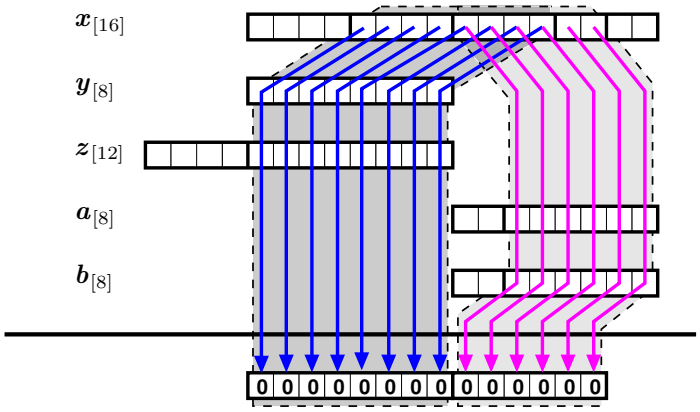


Figure 6.3: Data Dependencies of $\mathcal{F}_{[14]}$

Although $\mathcal{F}_{[14]}$ is not bitwise, we can observe that the data flow of $\mathcal{F}_{[14]}$ is at least bitwise on certain portions. According to (6.6) we have:

$$\mathcal{F}_{[14]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]})[5,0] \quad = \quad \mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]})$$

For all $\boldsymbol{u}_{[6]}, \boldsymbol{v}_{[6]}, \boldsymbol{w}_{[6]} \in \mathbb{B}_{[6]}$, let $\mathcal{B}^1_{[6]} : \mathbb{B}_{[6]} \times \mathbb{B}_{[6]} \times \mathbb{B}_{[6]} \longrightarrow \mathbb{B}_{[6]}$ be defined by

$$\mathcal{B}^1_{[6]}(\boldsymbol{u}_{[6]}, \boldsymbol{v}_{[6]}, \boldsymbol{w}_{[6]}) \;:=\; (\; \boldsymbol{u}_{[6]} \text{ or } \boldsymbol{v}_{[6]} \;) \text{ xor } \boldsymbol{w}_{[6]} \tag{6.7}$$

Then $\mathcal{B}^1_{[6]}$ is a bitwise bitvector function, and we have

$$\mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}) \quad = \quad \mathcal{B}^1_{[6]}(\boldsymbol{a}_{[8]}[5,0], \boldsymbol{b}_{[8]}[5,0], \boldsymbol{x}_{[16]}[7,2]) \tag{6.8}$$

as it is illustrated in Figure 6.4 below.

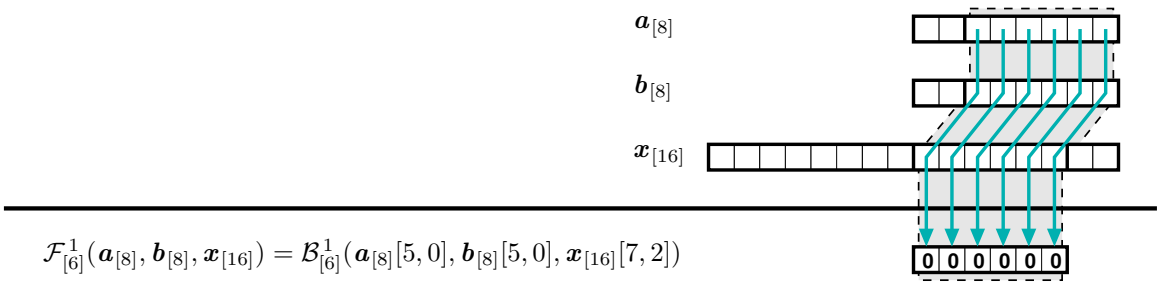


Figure 6.4: Partially Bitwise Data Dependencies of $\mathcal{F}^1_{[6]}$

We say that bitvector function $\mathcal{F}^1_{[6]}$ is partially bitwise with respect to $\boldsymbol{a}_{[8]}[5,0]$, $\boldsymbol{b}_{[8]}[5,0]$ and $\boldsymbol{x}_{[16]}[7,2]$. Satisfiability of bitvector equation e_1 can be characterized by satisfiability of the bitwise **BvSAT** problem related to $\mathcal{B}^1_{[6]}$, i.e. we have:

$$\langle \boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]} \rangle \;\; \textit{satisfies} \;\; (6.2) \qquad \Longleftrightarrow \qquad \mathcal{B}^1_{[6]}(\boldsymbol{a}_{[8]}[5,0], \boldsymbol{b}_{[8]}[5,0], \boldsymbol{x}_{[16]}[7,2]) \;=\; 0_{[6]}$$

Equation (6.6) furthermore yields:

$$\mathcal{F}_{[14]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]})[13,6] \;\; = \;\; \mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]})$$

For all $\boldsymbol{u}_{[8]}, \boldsymbol{v}_{[8]}, \boldsymbol{w}_{[8]} \in \mathbb{B}_{[8]}$ let $\mathcal{B}^2_{[8]} : \mathbb{B}_{[8]} \times \mathbb{B}_{[8]} \times \mathbb{B}_{[8]} \longrightarrow \mathbb{B}_{[8]}$ be defined by:

$$\mathcal{B}^2_{[8]}(\boldsymbol{u}_{[8]}, \boldsymbol{v}_{[8]}, \boldsymbol{w}_{[8]}) \;:=\; (\; \boldsymbol{u}_{[8]} \;\mathsf{and}\; \boldsymbol{v}_{[8]} \;)\; \mathsf{xor}\; \boldsymbol{w}_{[8]} \tag{6.9}$$

Then $\mathcal{B}^2_{[8]}$ is a bitwise bitvector function for which the following holds:

$$\mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]})[7,0] \;\; = \;\; \mathcal{B}^2_{[8]}(\boldsymbol{x}_{[16]}[11,4], \boldsymbol{y}_{[12]}[7,0], \boldsymbol{z}_{[8]}[7,0]) \tag{6.10}$$

Hence, $\mathcal{F}^2_{[8]}$ is partially bitwise in $\boldsymbol{x}_{[16]}[11,4]$, $\boldsymbol{y}_{[12]}[7,0]$ and $\boldsymbol{z}_{[8]}[7,0]$, as shown in Figure 6.5.

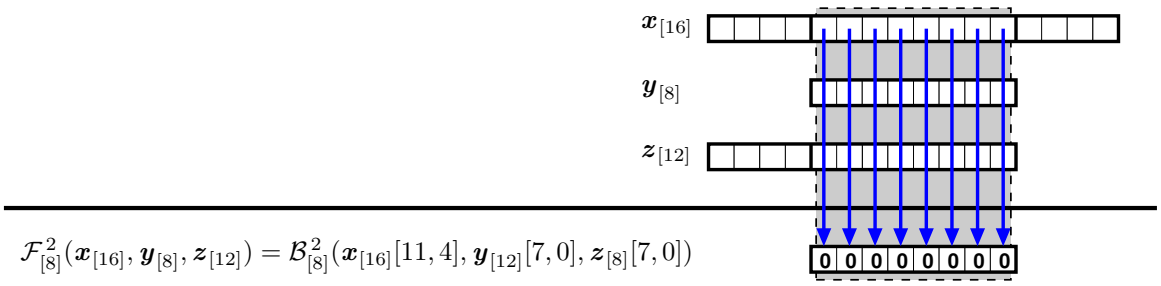


Figure 6.5: Partially Bitwise Data Dependencies of $\mathcal{F}^2_{[8]}$

As a consequence, satisfying solutions of bitvector equation e_2 can be characterized in the following way:

$$\langle \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]} \rangle \;\; \textit{satisfies} \;\; (6.4) \qquad \Longleftrightarrow \qquad \mathcal{B}^2_{[8]}(\boldsymbol{x}_{[16]}[11,4], \boldsymbol{y}_{[12]}[7,0], \boldsymbol{z}_{[8]}[7,0]) \;=\; 0_{[8]}$$

Note: For both bitvector equations e_1 and e_2, we have reduced satisfiability of a general **BvSAT** problem to satisfiability of a bitwise **BvSAT** problem. In order to do so, we have conceptually decomposed the bitvectors (bitvector variables) into contiguous parts of neighboring bits for which a bitwise data flow exists. Thus, for each equation the reduction technique which is presented in Chapter 4 can be applied individually.

We will now address satisfiability of E which can be characterized by conjunctive satisfiability of the two bitwise **BvSAT** problems associated with $\mathcal{B}^1_{[6]}$ and $\mathcal{B}^2_{[8]}$:

$$\begin{aligned}\langle \boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}\rangle \ \textit{satisfies}\ E \quad \Longleftrightarrow \\ \langle \boldsymbol{a}_{[8]}[5,0], \boldsymbol{b}_{[8]}[5,0], \boldsymbol{x}_{[16]}[7,2]\rangle \ \ \textit{satisfies}\ \ \mathbf{BvSAT}(\mathcal{B}^1_{[6]}) \\ \wedge\ \ \langle \boldsymbol{x}_{[16]}[11,4], \boldsymbol{y}_{[12]}[7,0], \boldsymbol{z}_{[8]}[7,0]\rangle \ \ \textit{satisfies}\ \ \mathbf{BvSAT}(\mathcal{B}^2_{[8]})\end{aligned} \tag{6.11}$$

As we observe, both bitwise **BvSAT** problems cannot be considered independently because both reference and share a common part of bits of bitvector variable $\boldsymbol{x}_{[16]}$, namely $\boldsymbol{x}_{[16]}[7,4]$ (see also Figure 6.3). Our primary objective is to find a decomposition of the initial **BvSAT** problem associated with E into a collection of unattached bitwise **BvSAT** problems, which then can be solved independently. In such a case, the presented reduction technique can be applied individually to each bitwise **BvSAT** problem. This objective can be achieved by a more detailed analysis of the bit-level data dependencies and by further refined decomposition of the bitvector variables. First, from (6.8) and (6.10), we conclude:

$$\begin{aligned}\mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]})[5,2] &= \mathcal{B}^1_{[6]}(\boldsymbol{a}_{[8]}[5,0], \boldsymbol{b}_{[8]}[5,0], \boldsymbol{x}_{[16]}[7,2])[5,2] \\ \mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]})[1,0] &= \mathcal{B}^1_{[6]}(\boldsymbol{a}_{[8]}[5,0], \boldsymbol{b}_{[8]}[5,0], \boldsymbol{x}_{[16]}[7,2])[1,0] \\ \mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]})[7,4] &= \mathcal{B}^2_{[8]}(\boldsymbol{x}_{[16]}[11,4], \boldsymbol{y}_{[12]}[7,0], \boldsymbol{z}_{[8]}[7,0])[7,4] \\ \mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]})[3,0] &= \mathcal{B}^2_{[8]}(\boldsymbol{x}_{[16]}[11,4], \boldsymbol{y}_{[12]}[7,0], \boldsymbol{z}_{[8]}[7,0])[3,0]\end{aligned} \tag{6.12}$$

As $\mathcal{B}^1_{[6]}$ and $\mathcal{B}^2_{[8]}$ are bitwise bitvector functions, we then have:

$$\begin{aligned}\mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]})[5,2] &= \mathcal{B}^1_{[4]}(\boldsymbol{a}_{[8]}[5,2], \boldsymbol{b}_{[8]}[5,2], \boldsymbol{x}_{[16]}[7,4]) \\ \mathcal{F}^1_{[6]}(\boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]})[1,0] &= \mathcal{B}^1_{[2]}(\boldsymbol{a}_{[8]}[1,0], \boldsymbol{b}_{[8]}[1,0], \boldsymbol{x}_{[16]}[3,2]) \\ \mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]})[7,4] &= \mathcal{B}^2_{[4]}(\boldsymbol{x}_{[16]}[11,8], \boldsymbol{y}_{[12]}[7,4], \boldsymbol{z}_{[8]}[7,4]) \\ \mathcal{F}^2_{[8]}(\boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]})[3,0] &= \mathcal{B}^2_{[4]}(\boldsymbol{x}_{[16]}[7,4], \boldsymbol{y}_{[12]}[3,0], \boldsymbol{z}_{[8]}[3,0])\end{aligned} \tag{6.13}$$

where $\mathcal{B}^2_{[4]} \simeq \mathcal{B}^2_{[8]}$ and $\mathcal{B}^1_{[4]} \simeq \mathcal{B}^1_{[6]}$ and $\mathcal{B}^1_{[2]} \simeq \mathcal{B}^1_{[6]}$ are the respective corresponding bitwise bitvector functions of reduced widths according to Definition 3.25. Then (6.11) can be rewritten as

$$\begin{aligned}\langle \boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]}\rangle \ \textit{satisfies}\ E \quad \Longleftrightarrow \\ \langle \boldsymbol{a}_{[8]}[5,2], \boldsymbol{b}_{[8]}[5,2], \boldsymbol{x}_{[16]}[7,4]\rangle \ \ \textit{satisfies}\ \ \mathbf{BvSAT}(\mathcal{B}^1_{[4]}) \\ \wedge\ \ \langle \boldsymbol{a}_{[8]}[1,0], \boldsymbol{b}_{[8]}[1,0], \boldsymbol{x}_{[16]}[3,2]\rangle \ \ \textit{satisfies}\ \ \mathbf{BvSAT}(\mathcal{B}^1_{[2]}) \\ \wedge\ \ \langle \boldsymbol{x}_{[16]}[11,8], \boldsymbol{y}_{[12]}[7,4], \boldsymbol{z}_{[8]}[7,4]\rangle \ \ \textit{satisfies}\ \ \mathbf{BvSAT}(\mathcal{B}^2_{[4]}) \\ \wedge\ \ \langle \boldsymbol{x}_{[16]}[7,4], \boldsymbol{y}_{[12]}[3,0], \boldsymbol{z}_{[8]}[3,0]\rangle \ \ \textit{satisfies}\ \ \mathbf{BvSAT}(\mathcal{B}^2_{[4]})\end{aligned} \tag{6.14}$$

Examining the sets of chunks occurring in the formula given above, we see that all **BvSAT** problems are mutually independent except for those associated with the first and the last set of chunks. Therefore, let $\mathcal{B}^3_{[4]} : \mathbb{B}_{[4]} \times \mathbb{B}_{[4]} \times \mathbb{B}_{[4]} \times \mathbb{B}_{[4]} \times \mathbb{B}_{[4]} \longrightarrow \mathbb{B}_{[4]}$ be defined by

$$\mathcal{B}^3_{[4]}(\boldsymbol{c}_{[4]}, \boldsymbol{d}_{[4]}, \boldsymbol{u}_{[4]}, \boldsymbol{v}_{[4]}, \boldsymbol{w}_{[4]}) \quad := \quad \mathcal{B}^1_{[4]}(\boldsymbol{c}_{[4]}, \boldsymbol{d}_{[4]}, \boldsymbol{u}_{[4]}) \text{ or } \mathcal{B}^2_{[4]}(\boldsymbol{u}_{[4]}, \boldsymbol{v}_{[4]}, \boldsymbol{w}_{[4]}) \tag{6.15}$$

for all $\boldsymbol{c}_{[4]}, \boldsymbol{d}_{[4]}, \boldsymbol{u}_{[4]}, \boldsymbol{v}_{[4]}, \boldsymbol{w}_{[4]} \in \mathbb{B}_{[4]}$. Then $\mathcal{B}^3_{[4]}$ is a bitwise bitvector function, and we have:

$$\begin{aligned} &\mathcal{B}^3_{[4]}(\boldsymbol{c}_{[4]}, \boldsymbol{d}_{[4]}, \boldsymbol{u}_{[4]}, \boldsymbol{v}_{[4]}, \boldsymbol{w}_{[4]}) = 0_{[4]} \quad \Longleftrightarrow \\ &\qquad\qquad \mathcal{B}^1_{[4]}(\boldsymbol{c}_{[4]}, \boldsymbol{d}_{[4]}, \boldsymbol{u}_{[4]}) = 0_{[4]} \quad \wedge \quad \mathcal{B}^2_{[4]}(\boldsymbol{u}_{[4]}, \boldsymbol{v}_{[4]}, \boldsymbol{w}_{[4]}) = 0_{[4]} \end{aligned}$$

Now, satisfiability of (6.1) can be characterized in the following way:

$$\begin{aligned} &\langle \boldsymbol{a}_{[8]}, \boldsymbol{b}_{[8]}, \boldsymbol{x}_{[16]}, \boldsymbol{y}_{[8]}, \boldsymbol{z}_{[12]} \rangle \textit{ satisfies } E \quad \Longleftrightarrow \\ &\qquad\qquad \langle \boldsymbol{a}_{[8]}[1,0], \boldsymbol{b}_{[8]}[1,0], \boldsymbol{x}_{[16]}[3,2] \rangle \textit{ satisfies } \textbf{BvSAT}(\mathcal{B}^1_{[2]}) \\ &\wedge \qquad \langle \boldsymbol{x}_{[16]}[11,8], \boldsymbol{y}_{[12]}[7,4], \boldsymbol{z}_{[8]}[7,4] \rangle \textit{ satisfies } \textbf{BvSAT}(\mathcal{B}^2_{[4]}) \\ &\wedge \quad \langle \boldsymbol{a}_{[8]}[5,2], \boldsymbol{b}_{[8]}[5,2], \boldsymbol{x}_{[16]}[7,4], \boldsymbol{y}_{[8]}[3,0], \boldsymbol{z}_{[12]}[3,0] \rangle \textit{ satisfies } \textbf{BvSAT}(\mathcal{B}^3_{[4]}) \end{aligned} \tag{6.16}$$

As can be seen, all three **BvSAT** problems described by $\mathcal{B}^1_{[4]}$, $\mathcal{B}^2_{[2]}$ and $\mathcal{B}^3_{[4]}$ now are independent because chunks of $\boldsymbol{a}_{[8]}$, $\boldsymbol{b}_{[8]}$, $\boldsymbol{x}_{[16]}$, $\boldsymbol{y}_{[8]}$, $\boldsymbol{z}_{[12]}$ which are referenced in different problems are disjoint, i.e. do not share any common bits. We call (6.16) a *bitwise decomposition* of E (or of $\mathcal{F}_{[14]}$, respectively).

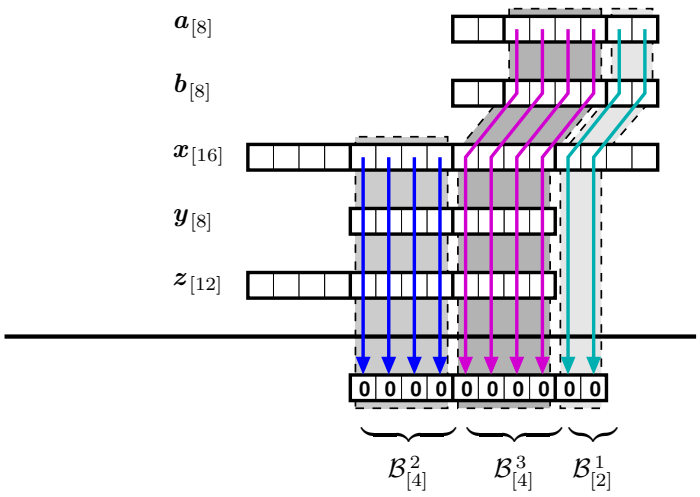


Figure 6.6: Bitwise Decomposition of (6.1)

Note: In order to reduce satisfiability of a general **BvSAT** problem which is related to a system of bitvector equations to conjunctive satisfiability of a collection of mutually independent bitwise **BvSAT** problems, we have done the following: we have identified sets of disjoint slices of the bitvectors (bitvector variables) such that for each set a bitwise bitvector function exists for which the corresponding bitwise **BvSAT** problem describes the set of all satisfying solutions of the system bitvector equations, restricted to the respective slices occurring in this set.

In the following sections we will formalize this decomposition technique and show how the decomposition of the bitvectors into slices of contiguous bits and the grouping according to bitwise data dependencies can be computed. We start by introducing some basic definitions in order to provide a formal representation for such bitwise decompositions.

6.2 Chunks of Bitvectors and Sets of Chunks

A bitvector $\boldsymbol{x}_{[n]} = \langle v_{n-1}, \ldots, v_1, v_0 \rangle \in \mathbb{B}_{[n]}$ is an array of bits which are ordered in a linear fashion. Neighboring bits are considered as *contiguous* with respect to this linear ordering. A contiguous slice $\langle v_j, v_{j-1}, \ldots, v_{i+1}, v_i \rangle$ of bits of $\boldsymbol{x}_{[n]}$ is called a *chunk* of $\boldsymbol{x}_{[n]}$.

Definition 6.1 (Chunk) *Let $n \in \mathbb{N}_+$ and let $\boldsymbol{x}_{[n]} \in \mathbb{B}_{[n]}$ be a bitvector of width n. Let $m \in \mathbb{N}_+$. A **chunk** of width m of $\boldsymbol{x}_{[n]}$ is a contiguous part of bits of $\boldsymbol{x}_{[n]}$, determined by $i, j \in \mathbb{N}_+$ with $0 \leq i \leq j < n$ and $j - i + 1 = m$ and the extraction expression $\boldsymbol{x}_{[n]}[j, i]$.* ■

In the following, we will always assume that a fixed and ordered number $k \in \mathbb{N}_+$ of bitvector widths $n_1, \ldots, n_k \in \mathbb{N}_+$ is given and that the notion of chunks always refers to the values of k bitvector variables $\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}$. A chunk always references a specific contiguous slice $\boldsymbol{x}^l_{[n_l]}[j, i]$ of bits of the value of one of these bitvectors, determined by $l \in \mathbb{N}_+$ and $i, j \in \mathbb{N}$. We introduce the following abstract formal notation for the specification of chunks.

Definition 6.2 (Representation of Chunks) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $l \in \{1, \ldots, k\}$ and $i, j \in \mathbb{N}$ with $0 \leq i \leq j < n_l$. Then the triple*

$$\langle l, j, i \rangle$$

represents the chunk $\boldsymbol{x}^l_{[n_l]}[j, i]$ for any associated ordered set of bitvectors $\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}$ of widths $n_1, \ldots, n_k$. ■

If the current framework, consisting of $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$, is obvious from the context, then such a triple $\langle l, j, i \rangle$ is shortly called a *chunk*. We furthermore introduce a notation for the set of all chunks referring to k and $n_1, \ldots, n_k$.

Definition 6.3 (Chunks) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Then let*

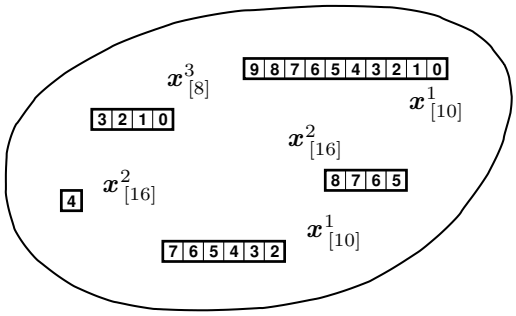
$$\mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle} := \{ \langle l, j, i \rangle \mid l \in \{1, \ldots, k\} \ \wedge \ i, j \in \mathbb{N}_+ \ \wedge \ 0 \leq i \leq j < n_l \}.$$

denote the set of all chunks referring to k ordered bitvectors of widths $n_1, \ldots, n_k$. ■

One of the main data types which will be used in the reasoning throughout the rest of this chapter are *sets of chunks*. We agree that the notion of sets of chunks always refers to subsets of chunks of a fixed set $\mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ which is assumed to be given as framework.

Definition 6.4 (Sets Of Chunks) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. A **set of chunks** is any subset $C \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$.* ■

Assuming that, within the given framework, $\mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ is associated with bitvector variables $\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}$, we will illustrate sets of chunks $C \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ in the way as it is shown in Figure 6.7.



$$\{\, \langle 1,9,0\rangle, \langle 1,7,2\rangle, \langle 2,4,4\rangle, \langle 2,5,8\rangle, \langle 3,3,0\rangle \,\} \subseteq \mathsf{Chunks}_{\langle 10,16,8\rangle}$$

Figure 6.7: Illustration of a Set of Chunks

Additionally, some further terminology for chunks and sets of chunks is defined. Two chunks are called *disjoint* if they do not reference any common bit of the associated bitvectors.

Definition 6.5 (Disjoint and Overlapping Chunks) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $\langle l_1, j_1, i_1\rangle, \langle l_2, j_2, i_2\rangle \in \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$. Then $\langle l_1, j_1, i_1\rangle$ and $\langle l_2, j_2, i_2\rangle$ are called **disjoint chunks**, if either $l_1 \neq l_2$, or if $i_1 > j_2$ or $i_2 > j_1$. Otherwise, $\langle l_1, j_1, i_1\rangle$ and $\langle l_2, j_2, i_2\rangle$ are named **overlapping chunks**.* ■

A set C of chunks is called *disjoint* if all chunks of C are pairwise disjoint, i.e. there does not exist any pair of different chunks of C which are overlapping.

Definition 6.6 (Disjoint Sets of Chunks) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$ and let $C \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ be a set of chunks. Then C is called **disjoint** if the following holds*

$$\langle l_1, j_1, i_1\rangle, \langle l_2, j_2, i_2\rangle \text{ overlapping} \quad \Longrightarrow \quad \langle l_1, j_1, i_1\rangle = \langle l_2, j_2, i_2\rangle$$

for all $\langle l_1, j_1, i_1\rangle, \langle l_2, j_2, i_2\rangle \in C$. ■

A collection of several sets of chunks is called *independent* if there do not exist any two chunks within any of the sets which reference the same bit of any of the bitvectors.

Definition 6.7 (Independent Sets of Chunks) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $q \in \mathbb{N}_+$ and let $C_1, \ldots, C_q \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ be sets of chunks. Then $C_1, \ldots, C_q$ are called **independent sets of chunks** if*

- $\forall\, i,j \in \{1,\ldots,q\} \;:\; (\, i \neq j \implies C_i \cap C_j = \emptyset\,)$, *and*
- $\bigcup_{i=1}^{q} C_i$ *is a disjoint set of chunks.*

■

In the following, we will mostly be concerned with a specific type of sets of chunks, namely the special case where all elements of a set of chunks are chunks of the same width. Such sets are called *homogeneous* sets of chunks.

Definition 6.8 (Homogeneous Sets of Chunks) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $C \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ be a set of chunks. Then C is called **homogeneous** if there exists $m \in \mathbb{N}_+$, such that for all $l \in \mathbb{N}_+$ and $i,j \in \mathbb{N}$ we have:*

$$\langle l,j,i\rangle \in C \quad \implies \quad j - i + 1 = m$$

*In such a case m is called the **width** of C and denoted by $\mathsf{width}(C) := m$.* ■

We also define a notation in order to indicate that all bits referenced by a specific chunk are also referenced by another chunk.

Definition 6.9 (Inclusion Relation for Chunks) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $\langle l_1,j_1,i_1\rangle, \langle l_2,j_2,i_2\rangle \in \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$. Then we define:*

$$\langle l_1,j_1,i_1\rangle \sqsubseteq \langle l_2,j_2,i_2\rangle \quad :\Longleftrightarrow \quad (\, l_1 = l_2 \;\wedge\; i_2 \leq i_1 \leq j_1 \leq j_2\,)$$

*If $\langle l_1,j_1,i_1\rangle \sqsubseteq \langle l_2,j_2,i_2\rangle$, then we say that $\langle l_2,j_2,i_2\rangle$ **comprises** $\langle l_1,j_1,i_1\rangle$.* ■

The same notation is used in order to indicate that a specific chunk is comprised by at least one chunk contained in a given set of chunks.

Definition 6.10 (Inclusion Relation for Chunks and Sets of Chunks) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $C \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ and let $\langle l,j,i\rangle \in \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$. Then we define:*

$$\langle l,j,i\rangle \sqsubseteq C \quad :\Longleftrightarrow \quad \exists\, \langle a,b,c\rangle \in C \;:\; \langle l,j,i\rangle \sqsubseteq \langle a,b,c\rangle$$

*If $\langle l,j,i\rangle \sqsubseteq C$, then we say that C **comprises** $\langle l,j,i\rangle$.* ■

A collection of several sets of chunks is called *complete* if each single-bit chunk of $\mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ is comprised by at least one of the sets of chunks.

Definition 6.11 (Complete Sets of Chunks) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $q \in \mathbb{N}_+$ and let $C_1, \ldots, C_q \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$. Then $C_1, \ldots, C_q$ are called **complete** if the following holds:*

$$\forall\ \langle l, i, i\rangle \in \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}\ \exists\ j \in \{1, \ldots, q\}\ :\ \langle l, i, i\rangle \sqsubseteq C_j \qquad \blacksquare$$

Finally, a strict ordering on chunks, i.e. a strict ordering on the set $\mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ of all triples representing chunks of bitvectors of given widths $n_1, \ldots, n_k$, is defined, which allows the elements of any set of chunks to be enumerated in a unique and ordered fashion.

Definition 6.12 (Strict Ordering on Chunks) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $\langle l_1, j_1, i_1\rangle, \langle l_2, j_2, i_2\rangle \in \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$. Then let the **strict ordering** $\ll$ be defined by*

$$\begin{aligned}\langle l_1, j_1, i_1\rangle \ll \langle l_2, j_2, i_2\rangle \quad :\Longleftrightarrow\\ (\, l_1 < l_2\,)\ \vee\ (\, l_1 = l_2\ \wedge\ i_1 < i_2\,)\ \vee\ (\, l_1 = l_2\ \wedge\ i_1 = i_2\ \wedge\ j_1 < j_2\,)\end{aligned} \qquad \blacksquare$$

6.3 Partially Bitwise Bitvector Functions

A given set of chunks $C \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ induces a multiple selection of chunks on any ordered choice of k bitvectors $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$. For each such set C we define a multiple extraction operation λ_C on the set $\mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]}$ of all k-tuples of bitvectors of widths $n_1, \ldots, n_k$. The selected chunks are ordered with respect to the strict ordering introduced in Definition 6.12.

Definition 6.13 (Chunk Selector Function) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $C \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ be a set of chunks and let $q := |C|$. Let $l_1, \ldots, l_q \in \{1, \ldots, k\}$ and $i_1, \ldots, i_q \in \mathbb{N}$ and $j_1, \ldots, j_q \in \mathbb{N}$, such that $C = \{\, \langle l_r, j_r, i_r\rangle \mid 1 \le r \le q\,\}$ and such that*

$$\langle l_1, j_1, i_1\rangle\ \ll\ \langle l_2, j_2, i_2\rangle\ \ll\ \ldots\ \ll\ \langle l_q, j_q, i_q\rangle\,.$$

*Let $m_r := j_r - i_r + 1$ for all $r \in \{1, \ldots, q\}$. Then the **chunk selector function***

$$\lambda_C : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[m_1]} \times \ldots \times \mathbb{B}_{[m_q]}$$

is defined by

$$\lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \quad := \quad \langle \boldsymbol{x}^{l_1}_{[n_{l_1}]}[j_1, i_1],\ \ldots\ , \boldsymbol{x}^{l_q}_{[n_{l_q}]}[j_q, i_q]\,\rangle$$

for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$. $\blacksquare$

We have now provided all the means which are necessary to formally characterize that a given bitvector function is partially bitwise as it has been informally described in the introductory example given in Section 6.1. In the following, let $\circ$ denote the composition of functions.

Definition 6.14 (Partially Bitwise Bitvector Functions) *Let $n \in \mathbb{N}_+$ and let $k \in \mathbb{N}_+$. Let $n_1, \ldots, n_k \in \mathbb{N}_+$ and let $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ be a k-ary bitvector function of width n on bitvectors of width $n_1, \ldots, n_k$. Let $i, j \in \mathbb{N}$ with $0 \leq i \leq j < n$, and let $m := j - i + 1$. Then $\mathcal{F}_{[n]}$ is **partially bitwise in** $[j, i]$ if there exists a set of chunks $C \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ and a bitwise bitvector function*

$$\mathcal{B}_{[m]} : \mathbb{B}_{[m]} \times \ldots \times \mathbb{B}_{[m]} \longrightarrow \mathbb{B}_{[m]}$$

such that $\mathcal{B}_{[m]} \circ \lambda_C$ is well defined and such that

$$\mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})[j, i] \quad = \quad \mathcal{B}_{[m]}\big(\lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})\big)$$

for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$, i.e. we have $\mathcal{F}_{[n]}[j, i] = \mathcal{B}_{[m]} \circ \lambda_C$. ■

Note: If a bitvector function $\mathcal{F}_{[n]}$ is partially bitwise in $[j, i]$ with respect to a set of chunks C and with respect to a bitwise bitvector function $\mathcal{B}_{[m]}$, then well-definedness of $\mathcal{B}_{[m]} \circ \lambda_C$ implies that C is a homogeneous set of chunks with $\mathsf{width}(C) = m$ and $|C| = q$, where $q \in \mathbb{N}_+$ is the arity of $\mathcal{B}_{[m]}$. Figure 6.8 illustrates the concept of bitvector functions which are partially bitwise on a set of chunks.

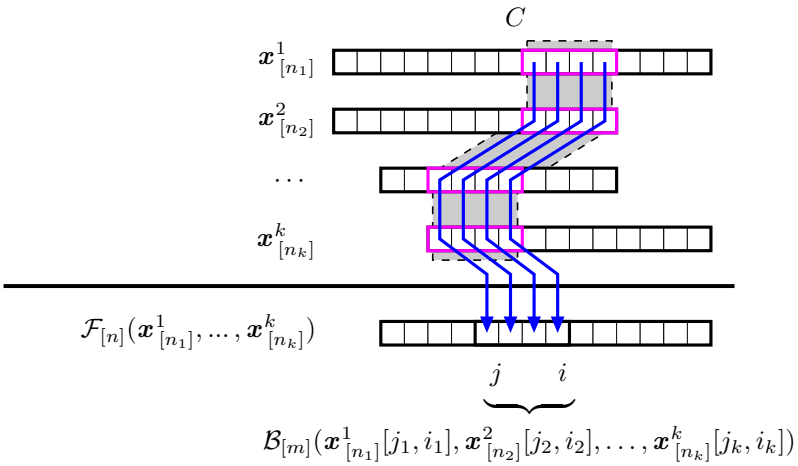


Figure 6.8: Partially Bitwise Bitvector Functions

Obviously, a bitvector function $\mathcal{F}_{[n]}$ which is partially bitwise in an interval $[j, i]$ is also partially bitwise in each sub interval of $[j, i]$.

Corollary 6.15 (Partially Bitwise Bitvector Functions) *Let $k \in \mathbb{N}_+$ and let $n \in \mathbb{N}_+$. Let $n_1, \ldots, n_k \in \mathbb{N}_+$ and let $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ be a bitvector function. Let $i, j \in \mathbb{N}$ with $0 \leq i \leq j < n$ and assume $\mathcal{F}_{[n]}$ to be partially bitwise in $[j, i]$. Let $i', j' \in \mathbb{N}$ with $i \leq i' \leq j' \leq j$. Then $\mathcal{F}_{[n]}$ is partially bitwise in $[j', i']$.* ■

Additionally, each bitvector function $\mathcal{F}_{[n]}$ is partially bitwise in each single-bit interval $[i, i]$ since the data dependencies for each bit of the result of $\mathcal{F}_{[n]}$ can always be described by a one-bit Boolean function, and since C can always be chosen as the set of all one-bit chunks belonging to $\mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$.

Corollary 6.16 (Partially Bitwise Bitvector Functions) *Let $k \in \mathbb{N}_+$ and let $n \in \mathbb{N}_+$. Let $n_1, \ldots, n_k \in \mathbb{N}_+$ and let $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ be a bitvector function. Let $i \in \mathbb{N}$ with $0 \leq i < n$. Then $\mathcal{F}_{[n]}$ is partially bitwise in $[i, i]$.* ■

We furthermore define the inverse operation of chunk selection which computes the inverse image of a collection of values of chunks with respect to a specific chunk selector function.

Definition 6.17 (Inversion of Chunk Selection) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $C \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ be a set of chunks and $q := |C|$. Let $l_1, \ldots, l_q \in \{1, \ldots, k\}$ and $i_1, \ldots, i_q \in \mathbb{N}$ and $j_1, \ldots, j_q \in \mathbb{N}$, such that $C = \{ \langle l_r, j_r, i_r \rangle \mid 1 \leq r \leq q \}$ and such that $\langle l_1, j_1, i_1 \rangle \ll \langle l_2, j_2, i_2 \rangle \ll \ldots \ll \langle l_q, j_q, i_q \rangle$. Let $m_r := j_r - i_r + 1$ for all $r \in \{1, \ldots, q\}$, and let $\boldsymbol{y}^1_{[m_1]} \in \mathbb{B}_{[m_1]}, \ldots, \boldsymbol{y}^q_{[m_q]} \in \mathbb{B}_{[m_q]}$. Then*

$$\lambda_C^{-1}(\boldsymbol{y}^1_{[m_1]}, \ldots, \boldsymbol{y}^q_{[m_q]}) \quad := \\ \{ \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \mid \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = \langle \boldsymbol{y}^1_{[m_1]}, \ldots, \boldsymbol{y}^q_{[m_q]} \rangle \}$$

*denotes the **inverse image** of $\boldsymbol{y}^1_{[m_1]}, \ldots, \boldsymbol{y}^q_{[m_q]}$ with respect to λ_C.* ■

6.4 Bitwise Decomposition

In the following, let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V := \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables. The objective of this chapter is to characterize satisfiability of a given system $E \subseteq \mathcal{E}(V)$ of bitvector equations by satisfiability of a collection of bitwise **BvSAT** problems. Our goal is to compute bitwise bitvector functions $\mathcal{B}^1_{[m_1]}, \ldots, \mathcal{B}^q_{[m_q]}$ with $q \in \mathbb{N}_+$ and $m_1, \ldots, m_q \in \mathbb{N}_+$ such that

$$E \text{ is satisfiable} \quad \Longleftrightarrow \quad \forall\, i \in \{1, \ldots, q\} \;:\; \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \text{ is satisfiable} \tag{6.17}$$

Moreover, it is desirable not only to just characterize *satisfiability* of E, but also to characterize *satisfying solutions* of E in a manner such that an easy way is provided to generate satisfying

solutions of E from satisfying solutions of the **BvSAT** problems, and vice versa. As we have seen, for each $E \subseteq \mathcal{E}(V)$, there exists a bitvector function $\mathcal{F}_{[n]} : \mathbb{B}_{[n_1]} \times \ldots \times \mathbb{B}_{[n_k]} \longrightarrow \mathbb{B}_{[n]}$ such that satisfying solutions of E are exactly the zeros of $\mathcal{F}_{[n]}$:

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \textit{ satisfies } E \quad \Longleftrightarrow \quad \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = \mathsf{0}_{[n]}$$

The first idea is to compute $[j_1, i_1], \ldots, [j_q, i_q]$ with $i_1, \ldots, i_q \in \mathbb{N}$ and $j_1, \ldots, j_q \in \mathbb{N}$ and bitwise bitvector functions $\mathcal{B}^1_{[m_1]}, \ldots, \mathcal{B}^q_{[m_q]}$ for $q \in \mathbb{N}_+$ and $m_1, \ldots, m_q \in \mathbb{N}_+$ and sets of chunks $C_1, \ldots, C_q$ such that $\mathcal{F}_{[n]}$ is partially bitwise in each $[j_s, i_s]$, i.e.

$$\forall\ s \in \{1, \ldots, q\}\ :\ \mathcal{F}_{[n]}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})[j_s, i_s]\ =\ \mathcal{B}^s_{[m_s]} \circ \lambda_{C_s}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$$

and such that (6.17) holds. The point is that, instead of characterizing satisfying solutions of E by zeros of $\mathcal{F}_{[n]}$, now satisfying solutions of E can be characterized by zeros of $\mathcal{B}^1_{[m_1]}, \ldots, \mathcal{B}^q_{[m_q]}$. This approach can be generalized as follows: given a system $E \subseteq \mathcal{E}(V)$ of bitvector equations our objective is to compute

- sets of chunks $C_1, \ldots, C_q \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ with $q \in \mathbb{N}_+$, and
- bitwise bitvector functions $\mathcal{B}^1_{[m_1]}, \ldots, \mathcal{B}^q_{[m_q]}$ with $m_1, \ldots, m_q \in \mathbb{N}_+$

such that $\mathcal{B}^i_{[m_i]} \circ \lambda_{C_i}$ is well defined for all $i \in \{1, \ldots, q\}$, and such that for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots,$ $\boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ we have:

$$\begin{aligned} &\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \textit{ satisfies } E \quad \Longleftrightarrow \\ &\qquad \forall\ i \in \{1, \ldots, q\}\ :\ \mathcal{B}^i_{[m_i]} \circ \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = \mathsf{0}_{[m_i]} \end{aligned} \tag{6.18}$$

If (6.18) holds, then we can conclude

$$E \textit{ is satisfiable} \quad \Longrightarrow \quad \forall\ i \in \{1, \ldots, q\}\ :\ \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \textit{ is satisfiable}$$

i.e. the forward implication of (6.17) holds. However, in general (6.18) is not sufficient to guarantee that the opposite implication of (6.17) also holds, as shown in the following example.

Example 6.18 (Decomposition into Bitwise BvSAT Problems) *Let $\boldsymbol{x}_{[3]}$ be a bitvector variable and consider the following bitvector equation $e \in \mathcal{E}(\{\boldsymbol{x}_{[3]}\})$:*

$$\big(\ \mathsf{neg}(\boldsymbol{x}_{[3]}[2,1])\ \mathsf{xor}\ \boldsymbol{x}_{[3]}[1,0]\ \big)\ \ \mathsf{or}\ \ \boldsymbol{x}_{[3]}[2,1] \quad = \quad \mathsf{0}_{[2]} \tag{6.19}$$

Let $\mathcal{F}_{[2]} : \mathbb{B}_{[3]} \longrightarrow \mathbb{B}_{[2]}$ be defined by $\boldsymbol{x}_{[3]} \mapsto (\mathsf{neg}(\boldsymbol{x}_{[3]}[2,1])\ \mathsf{xor}\ \boldsymbol{x}_{[3]}[1,0])\ \mathsf{or}\ \boldsymbol{x}_{[3]}[2,1]$ for all $\boldsymbol{x}_{[3]} \in \mathbb{B}_{[3]}$. Then we have:

$$\boldsymbol{x}_{[3]} \textit{ satisfies } e \quad \Longleftrightarrow \quad \mathcal{F}_{[2]}(\boldsymbol{x}_{[3]}) = \mathsf{0}_{[2]}$$

$\mathcal{F}_{[2]}$ is not a bitwise bitvector function. Let $C := \{\ \langle 1,1,0\rangle, \langle 1,2,1\rangle\ \}$. Then $C \subseteq \mathsf{Chunks}_{\langle 3\rangle}$. Let $\mathcal{B}_{[2]} : \mathbb{B}_{[2]} \times \mathbb{B}_{[2]} \longrightarrow \mathbb{B}_{[2]}$ be defined by

$$\langle \boldsymbol{a}_{[2]}, \boldsymbol{b}_{[2]} \rangle \mapsto (\mathsf{neg}(\boldsymbol{a}_{[2]})\ \mathsf{xor}\ \boldsymbol{b}_{[2]})\ \mathsf{or}\ \boldsymbol{a}_{[2]}$$

for all $\boldsymbol{a}_{[2]}, \boldsymbol{b}_{[2]} \in \mathbb{B}_{[2]}$*. Then* $\mathcal{B}_{[2]}$ *is a bitwise bitvector function,* $\mathcal{B}_{[2]} \circ \lambda_C$ *is well defined, and the following holds for all* $\boldsymbol{x}_{[3]} \in \mathbb{B}_{[3]}$*:*

$$\mathcal{F}_{[2]}(\boldsymbol{x}_{[3]}) \;=\; \mathcal{B}_{[2]} \circ \lambda_C(\boldsymbol{x}_{[3]})$$

and thus

$$\begin{aligned} \boldsymbol{x}_{[3]} \textit{ satisfies } e \quad &\iff \quad \mathcal{B}_{[2]} \circ \lambda_C(\boldsymbol{x}_{[3]}) = 0_{[2]} \\ &\iff \quad \lambda_C(\boldsymbol{x}_{[3]}) \textit{ satisfies } \textbf{BvSAT}(\mathcal{B}_{[2]}) \end{aligned}$$

and we can conclude

$$e \textit{ is satisfiable} \quad \Longrightarrow \quad \textbf{BvSAT}(\mathcal{B}_{[2]}) \textit{ is satisfiable}$$

However, the opposite implication is not true, i.e.

$$\textbf{BvSAT}(\mathcal{B}_{[2]}) \textit{ is satisfiable} \quad \not\Longrightarrow \quad e \textit{ is satisfiable}$$

since in this example e *is not satisfiable because the* or *operator occurring in (6.19) requires*

$$\begin{aligned} & \boldsymbol{x}_{[3]}[2,1] = \mathsf{00} \;\; \wedge \;\; \mathsf{neg}(\boldsymbol{x}_{[3]}[2,1]) \;\mathsf{xor}\; \boldsymbol{x}_{[3]}[1,0] = \mathsf{00} \\ \Longrightarrow \quad & \boldsymbol{x}_{[3]}[2,1] = \mathsf{00} \;\; \wedge \;\; \mathsf{neg}(\boldsymbol{x}_{[3]}[2,1]) = \boldsymbol{x}_{[3]}[1,0] \\ \Longrightarrow \quad & \boldsymbol{x}_{[3]}[2,2] = \boldsymbol{x}_{[3]}[1,1] \;\; \wedge \;\; \mathsf{neg}(\boldsymbol{x}_{[3]}[2,2]) = \boldsymbol{x}_{[3]}[1,1] \\ \Longrightarrow \quad & \boldsymbol{x}_{[3]}[2,2] = \mathsf{neg}(\boldsymbol{x}_{[3]}[2,2]) \;\; \lightning \end{aligned}$$

Yet, $\textbf{BvSAT}(\mathcal{B}_{[2]})$ *is satisfiable, for example by* $\langle \mathsf{00}, \mathsf{11} \rangle$*.* □

The reason why in this example satisfiability of the bitwise **BvSAT** problem does not imply satisfiability of e, is the fact that the two chunks of $\boldsymbol{x}_{[3]}$ which are specified by C are overlapping and thus both their values cannot be considered as independent of each other, as it is done when solving the **BvSAT** problem. Therefore, in addition to (6.18), we also require that

- $C_1, \ldots, C_q$ are *independent* sets of chunks, and
- for each $i \in \{1, \ldots, q\}$, C_i is a *disjoint* set of chunks

Let $k_1 := |C_1|, \ldots, k_q := |C_q|$, and let $i \in \{1, \ldots, q\}$. Then the following holds:

$$C_i \textit{ disjoint} \quad \Longrightarrow \quad \forall\, \boldsymbol{y}^1_{[m_i]}, \ldots, \boldsymbol{y}^{k_i}_{[m_i]} \in \mathbb{B}_{[m_i]} \;:\; \lambda_C^{-1}(\boldsymbol{y}^1_{[m_i]}, \ldots, \boldsymbol{y}^{k_i}_{[m_i]}) \neq \emptyset$$

and furthermore:

$$\begin{aligned} & C_1, \ldots, C_q \textit{ independent} \quad \Longrightarrow \\ & \forall\, \boldsymbol{y}^{1,1}_{[m_1]}, \ldots, \boldsymbol{y}^{1,k_1}_{[m_1]} \in \mathbb{B}_{[m_1]}, \;\ldots,\; \boldsymbol{y}^{q,1}_{[m_q]}, \ldots, \boldsymbol{y}^{q,k_q}_{[m_q]} \in \mathbb{B}_{[m_q]} \;:\; \Big(\bigcap_{i=1}^{q} \lambda_{C_i}^{-1}(\boldsymbol{y}^{i,1}_{[m_i]}, \ldots, \boldsymbol{y}^{i,k_i}_{[m_i]}) \Big) \neq \emptyset \end{aligned}$$

That is, if $C_1, \ldots, C_q$ are independent and disjoint sets of chunks, then we have:

$$\begin{aligned} & \forall\, \boldsymbol{y}^{1,1}_{[m_1]}, \ldots, \boldsymbol{y}^{1,k_1}_{[m_1]} \in \mathbb{B}_{[m_1]}, \;\ldots,\; \boldsymbol{y}^{q,1}_{[m_q]}, \ldots, \boldsymbol{y}^{q,k_q}_{[m_q]} \in \mathbb{B}_{[m_q]} \;: \\ & \qquad \exists\, \boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]} \;: \\ & \qquad\qquad \forall\, i \in \{1, \ldots, q\} \;:\; \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = \langle \boldsymbol{y}^{i,1}_{[m_i]}, \ldots, \boldsymbol{y}^{i,k_i}_{[m_i]} \rangle \end{aligned} \tag{6.20}$$

and hence, for all $\boldsymbol{y}^{1,1}_{[m_1]}, \ldots, \boldsymbol{y}^{1,k_1}_{[m_1]} \in \mathbb{B}_{[m_1]}, \ldots, \boldsymbol{y}^{q,1}_{[m_q]}, \ldots, \boldsymbol{y}^{q,k_q}_{[m_q]} \in \mathbb{B}_{[m_q]}$, we have

$$\begin{aligned} \forall\, i \in \{1,\ldots,q\} \;:\; & \langle \boldsymbol{y}^{i,1}_{[m_i]}, \ldots, \boldsymbol{y}^{i,k_i}_{[m_i]} \rangle \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \\ \Longrightarrow \quad & \exists\, \boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]} \;: \\ & \forall\, i \in \{1,\ldots,q\} \;:\; \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \end{aligned} \tag{6.21}$$

Together (6.18) and (6.21) imply:

$$\forall\, i \in \{1,\ldots,q\} \;:\; \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \textit{ is satisfiable} \quad\Longrightarrow\quad E \textit{ is satisfiable}$$

To summarize, our objective is to compute independent and disjoint sets of chunks $C_1, \ldots, C_q$ and bitwise bitvector functions $\mathcal{B}^1_{[m_1]}, \ldots, \mathcal{B}^1_{[m_q]}$ such that (6.18) holds. Then each satisfying solution of E induces a satisfying solution for each bitwise $\mathbf{BvSAT}(\mathcal{B}^i_{[m_i]})$ problem, and, contrariwise, for each collection of satisfying solutions of the $\mathbf{BvSAT}(\mathcal{B}^i_{[m_i]})$ problems a satisfying solution of E exists such that these solutions $\langle \boldsymbol{y}^{i,1}_{[m_{i,1}]}, \ldots, \boldsymbol{y}^{i,k_i}_{[n_{i,k_i}]} \rangle$ are exactly the results of chunk selection with respect to $C_1, \ldots, C_q$ when applied to the satisfying solutions of E. This conceptual approach is reflected by the following definition of a data structure which is called a *bitwise decomposition* of E and which provides a means to describe the set of satisfying solutions of E in terms of sets of chunks and bitwise bitvector functions.

Definition 6.19 (Bitwise Decomposition) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V := \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables and let $E \subseteq \mathcal{E}(V)$ be a system of bitvector equations. A **bitwise decomposition** $\mathcal{D}_E$ of E is a set*

$$\mathcal{D}_E \;=\; \{\, \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \,\}$$

with $q \in \mathbb{N}_+$ and $m_1, \ldots, m_q \in \mathbb{N}_+$, where

- *$C_1, \ldots, C_q \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k \rangle}$ are a complete selection of disjoint and independent sets of chunks, and*
- *$\mathcal{B}^1_{[m_1]}, \ldots, \mathcal{B}^q_{[m_q]}$ are bitwise bitvector functions, such that $\mathcal{B}^i_{[m_i]} \circ \lambda_{C_i}$ is well defined for each $i \in \{1,\ldots,q\}$,*

and such that the following holds for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$:

$$\begin{aligned} & \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \textit{ satisfies } E \quad\Longleftrightarrow \\ & \qquad \forall\, i \in \{1,\ldots,q\} \;:\; \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \end{aligned}$$

■

We will illustrate bitwise decompositions as annotated collections of sets of chunks as it is shown in Figure 6.9 below.

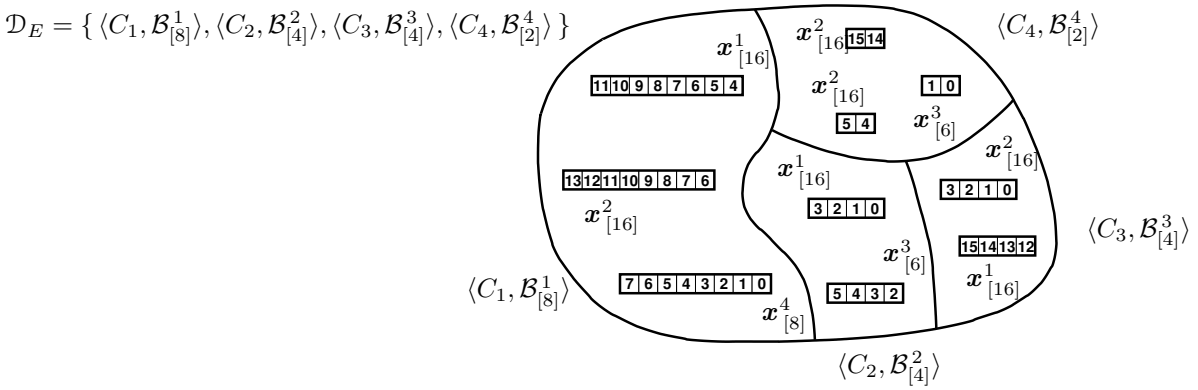


Figure 6.9: Illustration of Bitwise Decompositions

Corollary 6.16 implies that for each system of bitvector equations there always exists at least one bitwise decomposition because it is always possible to describe satisfiability of a system of bitvector equations by a Boolean formula involving Boolean variables for all single-bit chunks of the bitvector variables.

Corollary 6.20 (Existence of Bitwise Decompositions) *Let E be a system of bitvector equations. Then there exists a bitwise decomposition of E.* ■

Furthermore, Corollary 6.15 implies that in general bitwise decompositions of systems of bitvector equations are ambiguous, i.e. usually there exists a multitude of equivalent bitwise decompositions of the same systems of bitvector equations because the bitwise satisfiability problem corresponding to each $\langle C_i, \mathcal{B}^i_{[m_i]} \rangle$ can always be refined into an equivalent conjunction of bitwise satisfiability problems of smaller widths.

Corollary 6.21 (Ambiguity of Bitwise Decompositions) *Let E be a system of bitvector equations and let $\mathcal{D}_E$ be a bitwise decomposition of E which contains at least one set of chunks C with* $\mathsf{width}(C) > 1$. *Then each* ***refinement*** *of $\mathcal{D}_E$ is also a bitwise decomposition of E.* ■

The notion of *refinements* of bitwise decompositions of systems of bitvector equations is explained in detail in Section 6.6.3.

6.5 Computing the Decomposition

Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V := \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables, and let $E \subseteq \mathcal{E}(V)$ be a system of bitvector equations. We will incrementally compute a bitwise decomposition $\mathcal{D}_E$ of E by induction on E. It is presumed that all equations $e \in E$ are given in a

specific *normalized* form, as stated in the assumption (6.22) of the induction step. In Chapter 7, normalization of bitvector equations will be explained, and we will show that this assumption can be made without loss of generality.

For the induction base consider an empty set of bitvector equations $E = \emptyset$. Then a bitwise decomposition of E exists for trivial reasons. For each $i \in \{1, \ldots k\}$ define

- $C_i := \{ \langle i, n_i - 1, 0 \rangle \}$, and
- $\mathcal{B}^i_{[n_i]} : \mathbb{B}_{[n_i]} \longrightarrow \mathbb{B}_{[n_i]}$ with $\boldsymbol{x}_{[n_i]} \mapsto 0_{[n_i]}$ for all $\boldsymbol{x}_{[n_i]} \in \mathbb{B}_{[n_i]}$

and let $\mathcal{D}_E := \{ \langle C_1, \mathcal{B}^1_{[n_1]} \rangle, \ldots, \langle C_k, \mathcal{B}^k_{[n_k]} \rangle \}$. Then $\mathcal{D}_E$ is a bitwise decomposition of E.

For the induction step it is assumed that a system $E \subseteq \mathcal{E}(V)$ of bitvector equations and a bitwise decomposition $\mathcal{D}_E$ of E and an additional bitvector equation $e \in \mathcal{E}(V)$ are given. We show how to obtain a bitwise decomposition of the augmented system $E' := E \cup \{ e \}$.

Let $q \in \mathbb{N}_+$ and $m_1, \ldots, m_q \in \mathbb{N}_+$. Let $C_1, \ldots, C_q \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ be sets of chunks and $\mathcal{B}^1_{[m_1]}, \ldots, \mathcal{B}^q_{[m_q]}$ be bitwise bitvector functions such that

$$\mathcal{D}_E = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$$

is the bitwise decomposition of E.

Assume that furthermore a set of chunks $C \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ and a bitwise bitvector function $\mathcal{B}_{[m]}$ with $m \in \mathbb{N}_+$ are given such that $\mathcal{B}_{[m]} \circ \lambda_C$ is well defined and such that

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \textit{ satisfies } e \quad \Longleftrightarrow \quad \mathcal{B}_{[m]} \circ \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[m]} \tag{6.22}$$

holds for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$.

In the following sections we will present a procedure $\mathsf{Slice}(\mathcal{D}_E, \langle C, \mathcal{B}_{[m]} \rangle)$ which computes $p \in \mathbb{N}_+$ and $w_1, \ldots, w_p \in \mathbb{N}_+$ and sets of chunks $D_1, \ldots, D_p \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ and bitwise bitvector functions $\mathcal{H}^1_{[w_1]}, \ldots, \mathcal{H}^p_{[w_p]}$ such that

$$\mathcal{D}'_{E'} := \{ \langle D_1, \mathcal{H}^1_{[w_1]} \rangle, \ldots, \langle D_p, \mathcal{H}^p_{[w_p]} \rangle \}$$

is a bitwise decomposition of $E \cup \{ e \} = E'$.

$\mathcal{D}'_{E'}$ is computed by incrementally modifying $\mathcal{D}_E$. The Slice procedure is presented in Section 6.7. Slice applies several atomic modification steps to bitwise decompositions which are defined in the following section.

6.6 Modifying the Decomposition

In this section, several atomic procedures are defined which operate on and modify the decomposition data structures which have been introduced in Definition 6.19. These procedures establish basic transformation steps on bitwise decompositions and will be used as macros in the Slice procedure and in the main algorithms which compute a bitwise decomposition for a given system of bitvector equations.

6.6.1 Find

Let $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ be a bitwise decomposition as defined in Definition 6.19. The procedure $\mathsf{Find}(\mathcal{D}, \langle l, i \rangle)$, which is presented below, then returns the uniquely determined pair $\langle C_p, \mathcal{B}^p_{[m_p]} \rangle \in \mathcal{D}$ with $p \in \{1, \ldots, q\}$ for which the one-bit chunk $\langle l, i, i \rangle \in \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ with $l \in \{1, \ldots, k\}$ and $0 \leq i < n_l$ is comprised in the set of chunks C_p.

Algorithm 1 Find

```
1  Find(D, ⟨l, i⟩) {
2    let p := j ∈ {1, ..., q} with ⟨l, i, i⟩ ⊑ C_j        /* note: p is uniquely determined */
3    return ⟨C_p, B^p_[m_p]⟩
4  }
```

Reconsider the bitwise decomposition shown as an example in Figure 6.9. A sample Find operation for the same decomposition is illustrated in the following figure below.

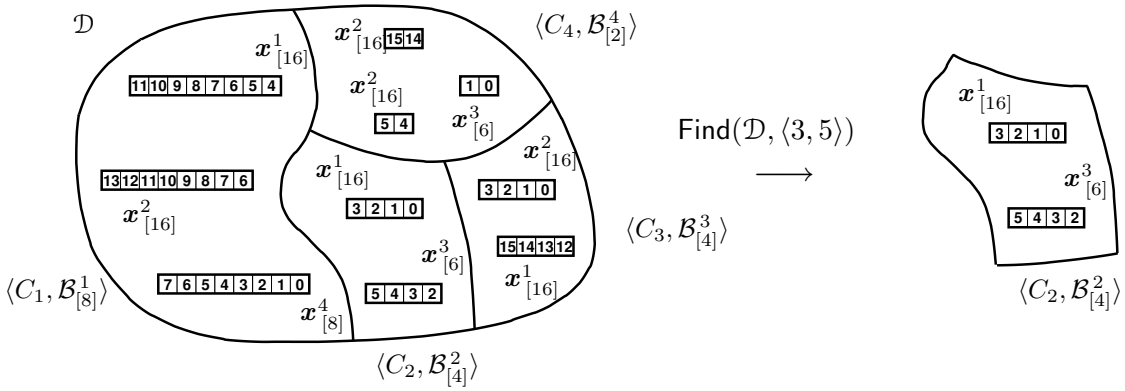


Figure 6.10: Illustration of Find Operations

Note: The structure $\mathcal{D}$ is not altered by Find operations, whereas the next two procedures perform modifying operations $\mathcal{D} \mapsto \mathcal{D}'$ on bitwise decompositions.

6.6.2 Union

Let $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ be a bitwise decomposition. The Union procedure computes the union of two pairs $\langle C_i, \mathcal{B}^i_{[m_i]} \rangle \in \mathcal{D}$ and $\langle C_j, \mathcal{B}^j_{[m_j]} \rangle \in \mathcal{D}$ for $i, j \in \{1, \ldots, q\}$. As a precondition, it is required that both sets C_i and C_j are sets of chunks of the same width, i.e. $m_i = m_j$, and thus both bitwise bitvector functions $\mathcal{B}^i_{[m_i]}$ and $\mathcal{B}^j_{[m_j]}$ are bitvector functions of the same width. The union of $\langle C_i, \mathcal{B}^i_{[m_i]} \rangle$ and $\langle C_j, \mathcal{B}^j_{[m_j]} \rangle$ is then defined to be the pair consisting of the regular set union of C_i and C_j and of the *bitwise union* of the bitvector functions $\mathcal{B}^i_{[m_i]}$ and $\mathcal{B}^j_{[m_j]}$. Bitwise union is defined as stated in the following corollary.

Corollary 6.22 (Bitwise Union of Chunk Selector Functions) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $m \in \mathbb{N}_+$ and let $C_1, C_2 \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ be homogeneous sets of chunks with $\mathsf{width}(C_1) = m$ and $\mathsf{width}(C_2) = m$. Let $\mathcal{B}^1_{[m]}$ and $\mathcal{B}^2_{[m]}$ be bitwise bitvector functions of width m such that $\mathcal{B}^1_{[m]} \circ \lambda_{C_1}$ and $\mathcal{B}^2_{[m]} \circ \lambda_{C_2}$ are well defined. Then there exists a uniquely determined bitwise bitvector function $\mathcal{B}_{[m]}$ of width m such that*

- *$\mathcal{B}_{[m]} \circ \lambda_{C_1 \cup C_2}$ is well defined, and*
- *for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ we have*

$$\mathcal{B}_{[m]} \circ \lambda_{C_1 \cup C_2}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \quad = \quad \mathcal{B}^1_{[m]} \circ \lambda_{C_1}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \quad \mathsf{or} \quad \mathcal{B}^2_{[m]} \circ \lambda_{C_2}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$$

*$\mathcal{B}_{[m]}$ is called the **bitwise union** of $\mathcal{B}^1_{[m]}$ and $\mathcal{B}^2_{[m]}$ with respect to C_1, C_2 and will be denoted by $\mathsf{OR}(\langle C_1, \mathcal{B}^1_{[m]} \rangle, \langle C_2, \mathcal{B}^2_{[m]} \rangle)$.*

Proof: Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $m \in \mathbb{N}_+$ and let $C_1, C_2 \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ be homogeneous sets of chunks with $\mathsf{width}(C_1) = m$ and $\mathsf{width}(C_2) = m$. Let $\mathcal{B}^1_{[m]}$ and $\mathcal{B}^2_{[m]}$ be bitwise bitvector functions of width m such that $\mathcal{B}^1_{[m]} \circ \lambda_{C_1}$ and $\mathcal{B}^2_{[m]} \circ \lambda_{C_2}$ are well defined. Let $\alpha_1 : \{ 1, \ldots, |C_1| \} \longrightarrow C_1$ and $\alpha_2 : \{ 1, \ldots, |C_2| \} \longrightarrow C_2$ be ordered bijective enumerations of the elements of C_1 and C_2, respectively, such that

$$\forall\, i, j \in \{ 1, \ldots, |C_1| \} \;:\; i < j \implies \alpha_1(i) \ll \alpha_1(j)$$

and

$$\forall\, i, j \in \{ 1, \ldots, |C_2| \} \;:\; i < j \implies \alpha_2(i) \ll \alpha_2(j)$$

Let furthermore $\beta : \{ 1, \ldots, |C_1 \cup C_2| \} \longrightarrow C_1 \cup C_2$ be the ordered bijective enumeration of the elements of $C_1 \cup C_2$ with

$$\forall\, i, j \in \{ 1, \ldots, |C_1 \cup C_2| \} \;:\; i < j \implies \beta(i) \ll \beta(j)$$

Then define

$$\mathcal{H}^1_{[m]} : \underbrace{\mathbb{B}_{[m]} \times \ldots \times \mathbb{B}_{[m]}}_{|C_1 \cup C_2|} \longrightarrow \mathbb{B}_{[m]} \qquad \text{and} \qquad \mathcal{H}^2_{[m]} : \underbrace{\mathbb{B}_{[m]} \times \ldots \times \mathbb{B}_{[m]}}_{|C_1 \cup C_2|} \longrightarrow \mathbb{B}_{[m]}$$

by

$$\mathcal{H}^1_{[m]}(\boldsymbol{y}^1_{[m]},\ldots,\boldsymbol{y}^{|C_1\cup C_2|}_{[m]}) \;:=\; \mathcal{B}^1_{[m]}(\boldsymbol{y}^{\beta^{-1}\circ\alpha_1(1)}_{[m]},\boldsymbol{y}^{\beta^{-1}\circ\alpha_1(2)}_{[m]},\ldots,\boldsymbol{y}^{\beta^{-1}\circ\alpha_1(|C_1|)}_{[m]})$$

and

$$\mathcal{H}^2_{[m]}(\boldsymbol{y}^1_{[m]},\ldots,\boldsymbol{y}^{|C_1\cup C_2|}_{[m]}) \;:=\; \mathcal{B}^2_{[m]}(\boldsymbol{y}^{\beta^{-1}\circ\alpha_2(1)}_{[m]},\boldsymbol{y}^{\beta^{-1}\circ\alpha_2(2)}_{[m]},\ldots,\boldsymbol{y}^{\beta^{-1}\circ\alpha_2(|C_2|)}_{[m]})$$

for all $\boldsymbol{y}^1_{[m]},\ldots,\boldsymbol{y}^{|C_1\cup C_2|}_{[m]} \in \mathbb{B}_{[m]}$. Then $\mathcal{H}^1_{[m]}$ and $\mathcal{H}^2_{[m]}$ are bitwise bitvector functions. Let

$$\mathcal{B}_{[m]} : \underbrace{\mathbb{B}_{[m]}\times\ldots\times\mathbb{B}_{[m]}}_{|C_1\cup C_2|} \longrightarrow \mathbb{B}_{[m]}$$

be defined by

$$\mathcal{B}_{[m]}(\boldsymbol{y}^1_{[m]},\ldots,\boldsymbol{y}^{|C_1\cup C_2|}_{[m]}) \;:=\; \mathcal{B}^1_{[m]}(\boldsymbol{y}^1_{[m]},\ldots,\boldsymbol{y}^{|C_1\cup C_2|}_{[m]}) \;\mathsf{or}\; \mathcal{B}^2_{[m]}(\boldsymbol{y}^1_{[m]},\ldots,\boldsymbol{y}^{|C_1\cup C_2|}_{[m]})$$

Then $\mathcal{B}_{[m]}$ is a bitwise bitvector function, and we have

$$\mathcal{B}_{[m]}\circ\lambda_{C_1\cup C_2}(\boldsymbol{x}^1_{[n_1]},\ldots,\boldsymbol{x}^k_{[n_k]}) \;=\; \mathcal{B}^1_{[m]}\circ\lambda_{C_1}(\boldsymbol{x}^1_{[n_1]},\ldots,\boldsymbol{x}^k_{[n_k]}) \;\mathsf{or}\; \mathcal{B}^2_{[m]}\circ\lambda_{C_2}(\boldsymbol{x}^1_{[n_1]},\ldots,\boldsymbol{x}^k_{[n_k]})$$

for all $\boldsymbol{x}^1_{[n_1]}\in\mathbb{B}_{[n_1]},\ldots,\boldsymbol{x}^k_{[n_k]}\in\mathbb{B}_{[n_k]}$. ■

Note: Depending on the sizes of the sets of chunks C_1 and C_2, $\mathcal{B}^1_{[m]}$ and $\mathcal{B}^2_{[m]}$ in general can be bitwise bitvector functions of different arities.

Example 6.23 (Bitwise Union of Chunk Selector Functions) *Let* $C_1 \subseteq \mathsf{Chunks}_{\langle 8,4,10\rangle}$ *and* $C_2 \subseteq \mathsf{Chunks}_{\langle 8,4,10\rangle}$ *with* $C_1 = \{\langle 1,7,4\rangle,\langle 2,3,0\rangle,\langle 3,10,7\rangle\}$ *and* $C_2 = \{\langle 1,3,0\rangle,\langle 3,10,7\rangle\}$. *Let* $\mathcal{B}^1_{[4]} : \mathbb{B}_{[4]}\times\mathbb{B}_{[4]}\times\mathbb{B}_{[4]} \longrightarrow \mathbb{B}_{[4]}$ *and* $\mathcal{B}^2_{[4]} : \mathbb{B}_{[4]}\times\mathbb{B}_{[4]} \longrightarrow \mathbb{B}_{[4]}$ *be bitwise bitvector functions. Then*

$$\mathcal{B}^1_{[4]}\circ\lambda_{C_1} : \mathbb{B}_{[8]}\times\mathbb{B}_{[4]}\times\mathbb{B}_{[10]} \longrightarrow \mathbb{B}_{[4]},\quad \langle \boldsymbol{x}^1_{[8]},\boldsymbol{x}^2_{[4]},\boldsymbol{x}^3_{[10]}\rangle \;\mapsto\; \mathcal{B}^1_{[4]}(\boldsymbol{x}^1_{[8]}[7,4],\boldsymbol{x}^2_{[4]},\boldsymbol{x}^3_{[10]}[10,7])$$

and

$$\mathcal{B}^2_{[4]}\circ\lambda_{C_2} : \mathbb{B}_{[8]}\times\mathbb{B}_{[4]}\times\mathbb{B}_{[10]} \longrightarrow \mathbb{B}_{[4]},\quad \langle \boldsymbol{x}^1_{[8]},\boldsymbol{x}^2_{[4]},\boldsymbol{x}^3_{[10]}\rangle \;\mapsto\; \mathcal{B}^2_{[4]}(\boldsymbol{x}^1_{[8]}[3,0],\boldsymbol{x}^3_{[10]}[10,7])$$

Let $\mathcal{B}_{[4]} := \mathsf{OR}(\langle C_1,\mathcal{B}^1_{[4]}\rangle,\langle C_2,\mathcal{B}^2_{[4]}\rangle)$. *Then we have*

$$\mathcal{B}_{[4]} : \mathbb{B}_{[4]}\times\mathbb{B}_{[4]}\times\mathbb{B}_{[4]}\times\mathbb{B}_{[4]} \longrightarrow \mathbb{B}_{[4]},\quad \langle \boldsymbol{y}^1_{[4]},\boldsymbol{y}^2_{[4]},\boldsymbol{y}^3_{[4]},\boldsymbol{y}^4_{[4]}\rangle \;\mapsto\; \mathcal{B}^1_{[4]}(\boldsymbol{y}^2_{[4]},\boldsymbol{y}^3_{[4]},\boldsymbol{y}^4_{[4]}) \;\mathsf{or}\; \mathcal{B}^2_{[4]}(\boldsymbol{y}^1_{[4]},\boldsymbol{y}^4_{[4]})$$

and

$$\mathcal{B}_{[4]}\circ\lambda_{C_1\cup C_2} : \mathbb{B}_{[8]}\times\mathbb{B}_{[4]}\times\mathbb{B}_{[10]} \longrightarrow \mathbb{B}_{[4]},\quad \langle \boldsymbol{x}^1_{[8]},\boldsymbol{x}^2_{[4]},\boldsymbol{x}^3_{[10]}\rangle \;\mapsto\; \mathcal{B}_{[4]}(\boldsymbol{x}^1_{[8]}[3,0],\boldsymbol{x}^1_{[8]}[7,4],\boldsymbol{x}^2_{[4]},\boldsymbol{x}^3_{[10]}[10,7])$$ □

The algorithm which computes union operations for bitwise decompositions is shown below. The two pairs of $\mathcal{D}$ which are to be united are referenced by specifying two one-bit chunks and then using the Find procedure which has been introduced in 6.6.1. The resulting decomposition $\mathcal{D}'$ is then constructed by replacing $\langle C_i, \mathcal{B}^i_{[m_i]} \rangle$ and $\langle C_j, \mathcal{B}^j_{[m_j]} \rangle$ by their union.

Algorithm 2 Union

```
1  Union(𝒟, ⟨l1, i1⟩, ⟨l2, i2⟩) {
2    ⟨D1, ℋ¹[m]⟩ := Find(𝒟, ⟨l1, i1⟩)
3    ⟨D2, ℋ²[m]⟩ := Find(𝒟, ⟨l2, i2⟩)
4    D := D1 ∪ D2                                  /* assert width(D1) = width(D2) */
5    ℋ[m] := OR( ⟨D1, ℋ¹[m]⟩, ⟨D2, ℋ²[m]⟩ )
6    𝒟 := 𝒟 \ { ⟨D1, ℋ¹[m]⟩, ⟨D2, ℋ²[m]⟩ }
7    𝒟 := 𝒟 ∪ { ⟨D, ℋ[m]⟩ }
8    return 𝒟;
9  }
```

Using the example already shown in Figures 6.9 and 6.10, union operations on bitwise decompositions are illustrated below.

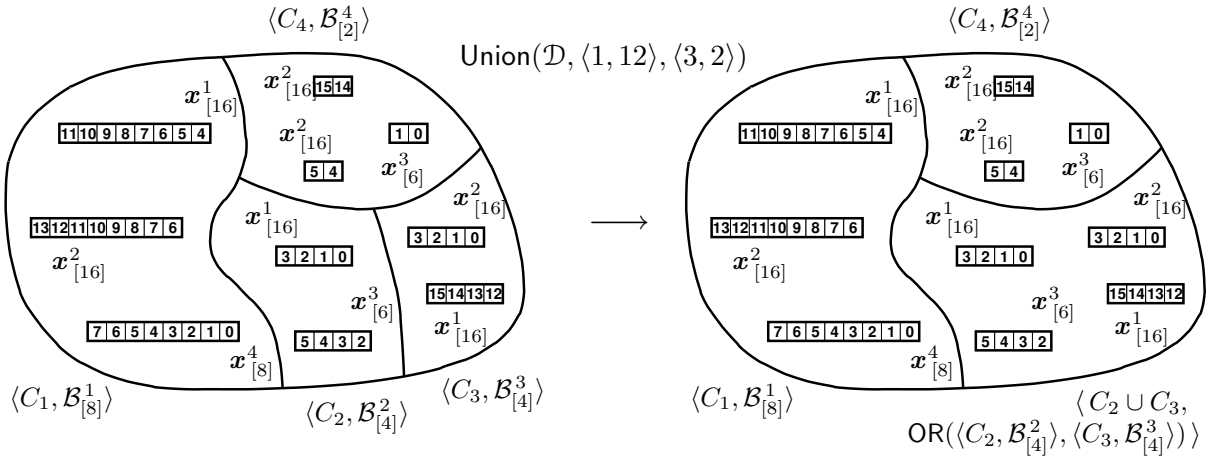


Figure 6.11: Illustration of Union Operations

Corollary 6.24 (Bitwise Compositions of Chunk Selector Functions) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $m \in \mathbb{N}_+$ and let $C_1, C_2 \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k \rangle}$ be homogeneous sets of chunks with $\mathsf{width}(C_1) = m$ and $\mathsf{width}(C_2) = m$. Let $\mathcal{B}^1_{[m]}$ and $\mathcal{B}^2_{[m]}$ be bitwise bitvector functions of width m such that $\mathcal{B}^1_{[m]} \circ \lambda_{C_1}$ and $\mathcal{B}^2_{[m]} \circ \lambda_{C_2}$ are well defined. Let $\mathcal{B}_{[m]} := \mathsf{OR}(\langle C_1, \mathcal{B}^1_{[m]} \rangle, \langle C_2, \mathcal{B}^2_{[m]} \rangle)$. Then for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ we have:*

$$\mathcal{B}_{[m]} \circ \lambda_{C_1 \cup C_2}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[m]} \iff \mathcal{B}^1_{[m]} \circ \lambda_{C_1}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[m]} \wedge \mathcal{B}^2_{[m]} \circ \lambda_{C_2}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[m]} \quad \blacksquare$$

Corollary 6.24 is an easy-to-see implication of the definition of bitwise unions of bitwise bitvector functions. As a consequence, the property of being a bitwise decomposition of a given system of bitvector equations is invariant under Union operations.

Theorem 6.25 (Bitwise Decompositions) *Let* $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ *be a bitwise decomposition. Let* $l_1, l_2 \in \{1, \ldots, k\}$ *and let* $i_1, i_2 \in \mathbb{N}$ *such that* $0 \leq i_1 < n_{l_1}$ *and* $0 \leq i_2 < n_{l_2}$*, and such that* $\mathsf{width}(C_{j_1}) = \mathsf{width}(C_{j_2})$ *for* $j_1, j_2 \in \mathbb{N}_+$ *with* $\langle l_1, i_1, i_1 \rangle \sqsubseteq C_{j_1}$ *and* $\langle l_2, i_2, i_2 \rangle \sqsubseteq C_{j_2}$*. Let*

$$\mathcal{D}' := \mathsf{Union}(\mathcal{D}, \langle l_1, i_1 \rangle, \langle l_2, i_2 \rangle)$$

and let E *be a system of bitvector equations. Then the following holds:*

$$\mathcal{D} \text{ is a bitwise decomposition of } E \quad \Longrightarrow \quad \mathcal{D}' \text{ is a bitwise decomposition of } E$$

Proof: The collection of sets of chunks belonging to $\mathcal{D}'$ is complete and independent because the sets belonging to $\mathcal{D}$ are complete and independent and Union computes the union of two homogeneous disjoint sets of chunks of the same width, which again results in a homogeneous disjoint set of chunks. Corollary 6.24 then yields that criterion (6.18) of bitwise decompositions, as stated in Definition 6.19, is preserved. ■

6.6.3 Split

The third procedure also performs a modifying operation $\mathcal{D} \mapsto \mathcal{D}'$ on bitwise decompositions $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ by splitting all chunks contained in a specific set of chunks C_j with $\langle C_j, \mathcal{B}^j_{[m_j]} \rangle \in \mathcal{D}$ at the same index position p. The resulting upper and lower parts of the chunks are grouped into two new sets of chunks C^{low} and C^{high}. The pair $\langle C_j, \mathcal{B}^j_{[m_j]} \rangle$ is removed from $\mathcal{D}$, and two new pairs consisting of C^{low} and C^{high} and the respective bitwise bitvector functions of reduced width corresponding to $\mathcal{B}^j_{[m_j]}$ are added.

Algorithm 3 Split

```
1   Split(D, ⟨l, i⟩, p) {
2     ⟨D, H_[m]⟩ := Find(D, ⟨l, i⟩)                    /* assert 0 < p < width(D) */
3     D_1 := { ⟨l, i + p − 1, i⟩ | l, j, i ∈ ℕ ∧ ⟨l, j, i⟩ ∈ D }
4     D_2 := { ⟨l, j, i + p⟩ | l, j, i ∈ ℕ ∧ ⟨l, j, i⟩ ∈ D }
5     m_1 := width(D_1)
6     m_2 := width(D_2)
7     H^1_[m_1] := H_[m_1]                             /* note: H^1_[m_1] ≃ H_[m] */
8     H^2_[m_2] := H_[m_2]                             /* note: H^2_[m_2] ≃ H_[m] */
9     D := D \ { ⟨D, H_[m]⟩ }
10    D := D ∪ { ⟨D_1, H^1_[m_1]⟩, ⟨D_2, H^2_[m_2]⟩ }
11    return D;
12  }
```

The set C_j which is to be split is referenced by specifying a one-bit chunk of $\mathsf{Chunks}_{\langle n_1,\ldots,n_k\rangle}$ and then calling Find. The third parameter of the procedure indicates the offset p at which the chunks of C_j are to be split. Figure 6.12 shows an example which modifies the bitwise decomposition which has already been used in the last sections and splits all chunks of the set containing bit 4 of $\boldsymbol{x}^1_{[16]}$ at offset position $p = 2$.

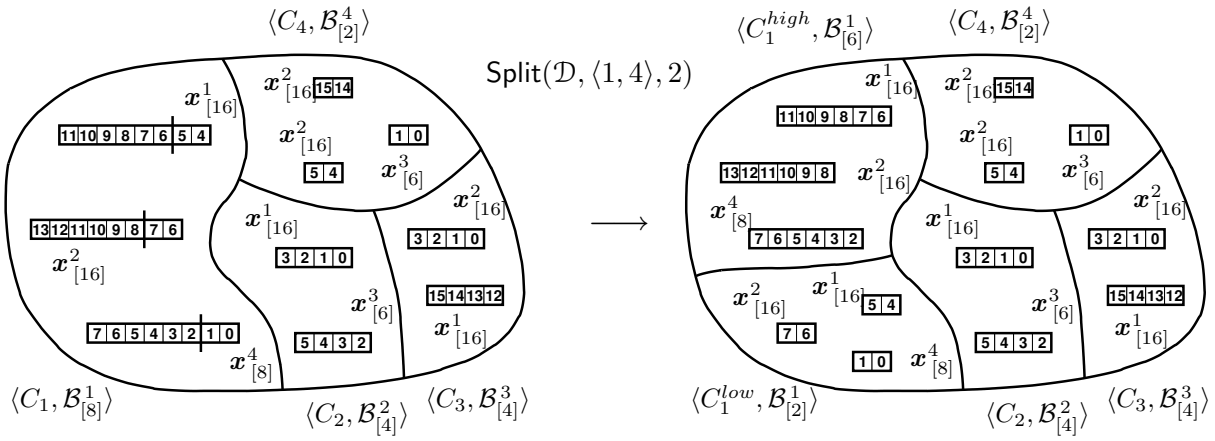


Figure 6.12: Illustration of Split Operations

Split operations on a bitwise decomposition of a given system E of bitvector equations again yield bitwise decompositions of E.

Theorem 6.26 (Bitwise Decompositions) *Let* $\mathcal{D} = \{\langle C_1, \mathcal{B}^1_{[m_1]}\rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]}\rangle\}$ *be a bitwise decomposition. Let* $l \in \{1, \ldots, k\}$ *and let* $i \in \mathbb{N}$ *such that* $0 \leq i < n_l$. *Let* $p \in \mathbb{N}_+$ *such that* $0 < p < \mathsf{width}(C_j)$ *for* $j \in \mathbb{N}_+$ *with* $\langle l, i, i\rangle \sqsubseteq C_j$. *Let*

$$\mathcal{D}' := \mathsf{Split}(\mathcal{D}, \langle l, i\rangle, p)$$

and let E *be a system of bitvector equations. Then the following holds:*

$$\mathcal{D} \text{ is a bitwise decomposition of } E \quad \Longrightarrow \quad \mathcal{D}' \text{ is a bitwise decomposition of } E$$

Proof: Splitting all chunks of a homogeneous disjoint set of chunks at the same index position and then grouping the resulting chunks according to upper and lower parts yields two homogeneous and disjoint sets of chunks. Thus the collection of sets of chunks belonging to $\mathcal{D}'$ again is complete and independent. Theorem 4.8 yields that criterion (6.18) of bitwise decompositions continues to hold because $\mathcal{H}^1_{[m_1]}$, $\mathcal{H}^2_{[m_2]}$ and $\mathcal{H}_{[m]}$ are corresponding bitwise bitvector functions with the same characteristic Boolean function. ■

If a bitwise decomposition $\mathcal{D}'$ can be obtained from another bitwise decomposition $\mathcal{D}$ by repeated application of Split operations, then $\mathcal{D}'$ is called a *refinement* of $\mathcal{D}$.

6.7 The Central Procedure: Bitwise Slicing

Bitwise decompositions $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ hold several aspects of information. The sets of chunks $C_1, \ldots, C_q$ explicitly indicate the grouping of contiguous parts of the bitvectors according to bitwise data dependencies, and $\mathcal{B}^1_{[m_1]}, \ldots, \mathcal{B}^q_{[m_q]}$ are a detailed specification of the bitwise data flow. For each bitvector variable $\boldsymbol{x}^l_{[n_l]}$ a bitwise decomposition $\mathcal{D}$ furthermore implicitly contains the information how $\boldsymbol{x}^l_{[n_l]}$ is decomposed into a complete set of disjoint chunks induced by the sets $C_1, \ldots, C_q$. Such a decomposition of a bitvector variable is called a *granularity* and is defined as follows.

Definition 6.27 (Granularity) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $l \in \{1, \ldots, k\}$. A* ***granularity*** *of $\boldsymbol{x}^l_{[n_l]} \in \mathbb{B}_{[n_l]}$ is a disjoint set of chunks*

$$G \quad \subseteq \quad (\, \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle} \, \cap \, \{ \, l \, \} \times \mathbb{N} \times \mathbb{N} \,)$$

such that for all $i \in \{0, \ldots, n_l - 1\}$ we have $\langle l, i, i \rangle \sqsubseteq G$. ■

Note: If $G \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$ is a granularity of $\boldsymbol{x}^l_{[n_l]}$, then there exist $q \in \mathbb{N}_+$ and $i_1, \ldots, i_q \in \mathbb{N}$ and $j_1, \ldots, j_q \in \mathbb{N}_+$ such that $G = \{ \langle l, j_1, i_1 \rangle, \ldots, \langle l, j_q, i_q \rangle \}$ and such that the bitvector equation

$$\boldsymbol{x}^l_{[n_l]} \quad = \quad \boldsymbol{x}^l_{[n_l]}[j_q, i_q] \; \otimes \; \ldots \; \otimes \; \boldsymbol{x}^l_{[n_l]}[j_1, i_1]$$

is a tautology.

Example 6.28 (Granularity) *Let $G := \{ \langle 1, 3, 0 \rangle, \langle 1, 11, 4 \rangle, \langle 1, 15, 12 \rangle \}$. Then G is a granularity of $\boldsymbol{x}^1_{[16]}$ and the bitvector equation*

$$\boldsymbol{x}^1_{[16]} \quad = \quad \boldsymbol{x}^1_{[16]}[15, 12] \; \otimes \; \boldsymbol{x}^1_{[16]}[11, 4] \; \otimes \; \boldsymbol{x}^1_{[16]}[3, 0]$$

is a tautology. Granularities will be illustrated in the way as G is illustrated in Figure 6.13. □

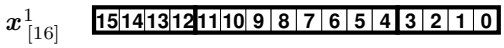


Figure 6.13: Granularity G of Example 6.28

The granularities which are induced by a bitwise decomposition then are given as follows.

Definition 6.29 (Granularities Induced by Bitwise Decompositions) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ be a bitwise decomposition with $C_1, \ldots, C_q \subseteq \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle}$. Let $l \in \{1, \ldots, k\}$. The* ***granularity of $\boldsymbol{x}^l_{[n_l]}$ induced by*** *$\mathcal{D}$ is given by*

$$\mathsf{gran}(\boldsymbol{x}^l_{[n_l]}, \mathcal{D}) \quad := \quad (\, \bigcup_{i=1}^{q} C_i \quad \cap \quad \{ \, l \, \} \times \mathbb{N} \times \mathbb{N} \,)$$

■

The definition of $\mathsf{gran}(\boldsymbol{x}^l_{[n_l]}, \mathcal{D})$ yields a granularity in the sense of Definition 6.27 because $C_1, \ldots, C_q$ is a complete and independent collection of disjoint sets of chunks. In Figure 6.14 the granularities are shown which are induced for all bitvector variables by the sample bitwise decomposition presented in Figure 6.9.

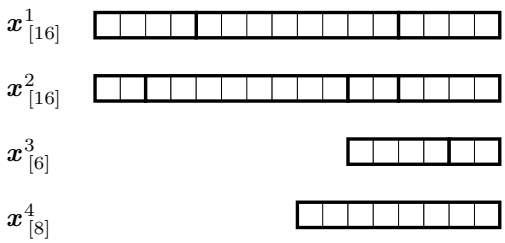


Figure 6.14: Granularities induced by the Bitwise Decomposition shown in Fig. 6.9

We will now present the central procedure Slice which computes the induction step as it has been announced in Section 6.5. The input parameters of the Slice procedure are a bitwise decomposition $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ and a set of chunks C and a bitwise bitvector function $\mathcal{B}_{[m]}$ such that $\mathcal{B}_{[m]} \circ \lambda_C$ is well defined. Slice incrementally modifies $\mathcal{D}$ such that after completion of the procedure the result is a bitwise decomposition $\mathcal{D}' = \{ \langle D_1, \mathcal{H}^1_{[w_1]} \rangle, \ldots, \langle D_p, \mathcal{H}^p_{[w_p]} \rangle \}$ such that for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ the following holds:

$$\begin{aligned} \forall\, i \in \{1, \ldots, p\} \;:\; & \lambda_{D_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{H}^i_{[w_i]}) \iff \\ & \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}_{[m]}) \;\wedge \\ & \forall\, i \in \{1, \ldots, q\} \;:\; \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \end{aligned} \tag{6.23}$$

The Slice procedure is shown in Algorithm 4. We illustrate the basic steps of Slice in Figures 6.15 to 6.21 using the sample bitwise decomposition which is shown in the following figure.

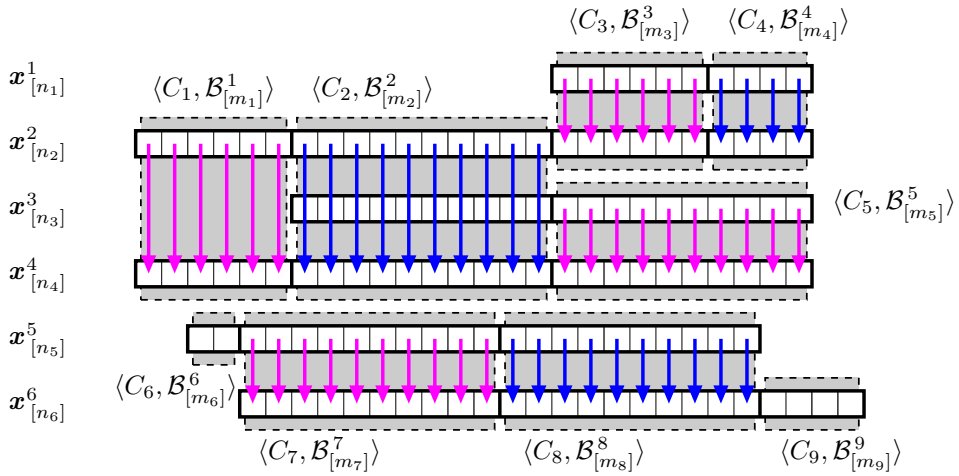


Figure 6.15: Sample Bitwise Decomposition $\mathcal{D}$

The granularities of $\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^6_{[n_6]}$ which are induced by the sets of chunks $C_1, \ldots, C_9$ of this decomposition are shown in the next figure.

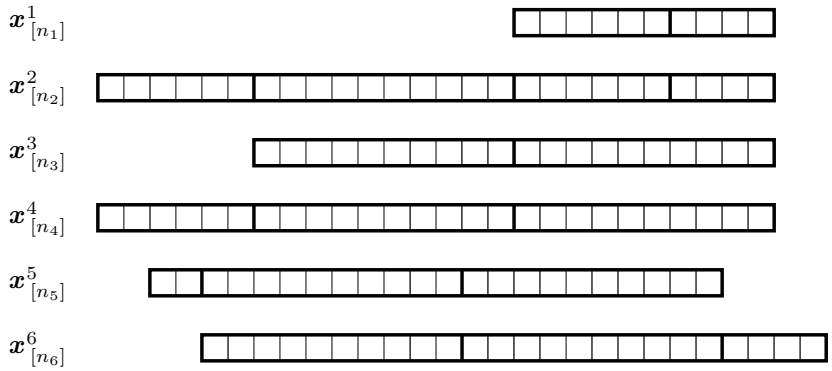


Figure 6.16: Granularities induced by $\mathcal{D}$

The bitwise bitvector function $\mathcal{B}_{[m]}$ and the set of chunks C impose additional bitwise data dependencies on $\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^6_{[n_6]}$, for example as it is shown below.

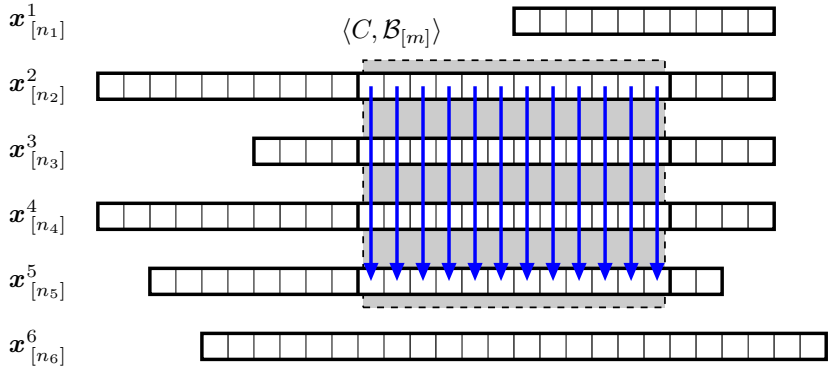


Figure 6.17: Sample Additional Bitwise Dependencies $\langle C, \mathcal{B}_{[m]} \rangle$

The task is to include these additional data dependencies in $\mathcal{D}$ in a way such that the result is a bitwise decomposition $\mathcal{D}'$ for which (6.23) holds. In doing so, the granularities of the bitvector variables $\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^6_{[n_6]}$, which are induced by the set of chunks C, have to be taken into account. These granularities are shown in Figure 6.18.

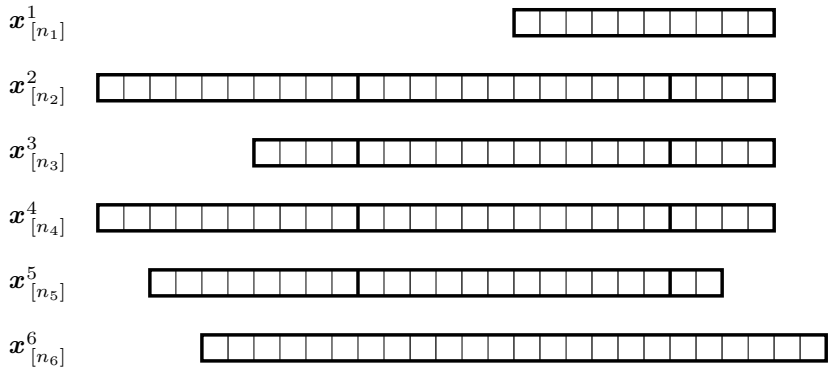


Figure 6.18: Granularities induced by $\langle C, \mathcal{B}_{[m]} \rangle$

In order to be able to combine the data dependencies which are shown in Figure 6.15 and in Figure 6.17 in a bitwise fashion, the Slice procedure first computes the common refinement of the two respective granularities illustrated in Figures 6.16 and 6.18. The common refinement is the superposition of both granularities for all chunks contained in C and for all chunks of $C_1, \ldots, C_9$ which are related to chunks of C by bitwise data dependencies.

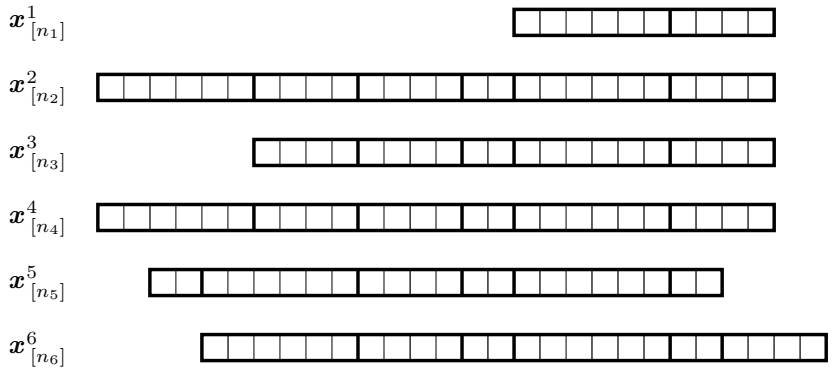


Figure 6.19: Superposition of the Granularities

The superposition is computed by first splitting all sets C_i of $\mathcal{D}$ which contain chunks which are overlapping with chunks of C at the respective index positions caused by the lower and upper bounds of the chunks of C. This is performed in steps 3–16 of Algorithm 4. As a next step, all chunks of C are split at all inner index positions caused by the bounds of the corresponding chunks contained in the sets C_i. This is done by traversing the bits of chunks of C from right to left and by computing the width of the shortest chunks as occurring in the sets C_i (steps 21–26 of Algorithm 4). Once the shortest width has been determined, all chunks of C are split at the respective index positions (steps 27–29 of Algorithm 4). The conceptual effect of these

splittings can be seen in Figure 6.20 which illustrates the resulting intermediate state of the bitwise decomposition $\mathcal{D}$.

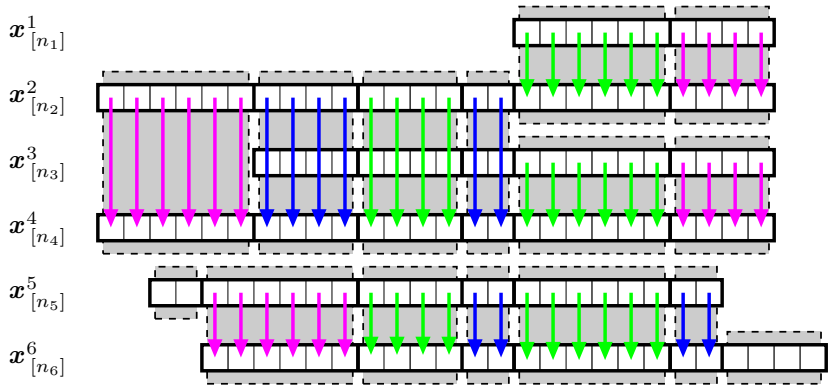


Figure 6.20: Splitting induced by $\mathcal{D}$ and $\langle C, \mathcal{B}_{[m]} \rangle$

The effect is that bits of chunks of C which are related to each other by the bitwise data dependencies specified by $\mathcal{B}_{[m]}$ are now contained in sets of chunks of the same width, and these sets now can be combined by union operations (steps 31–33 of Algorithm 4). The definition of bitwise unions of bitvector functions, as provided in Corollary 6.22, ensures that the bitwise data dependencies are combined conjunctively as required by Equation (6.23). Finally, in steps 35–38 of Algorithm 4, the bitwise dependencies of $\mathcal{B}_{[m]}$ are added. The resulting state of $\mathcal{D}$ is illustrated below.

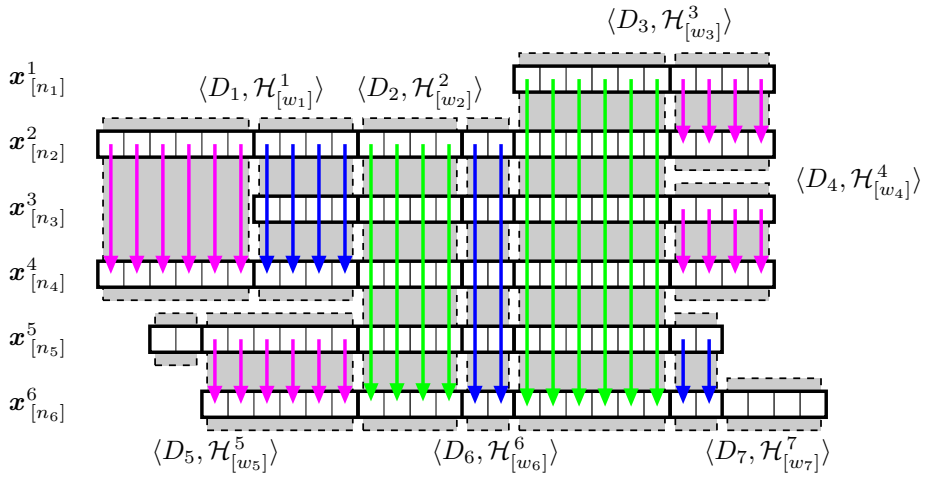


Figure 6.21: Resulting Bitwise Decomposition

Note: The conceptual steps of inner splitting of the chunks of C from left to right according to shortest width and the successive union of the respective resulting sets of chunks are illustrated separately as two subsequent steps in Figures 6.20 and 6.21. However, for reasons of efficiency, both steps can be nested as done in steps 19–41 of Algorithm 4.

Algorithm 4 Slicing

```
1  Slice(𝒟, ⟨C, ℬ_[m]⟩) {
2    let ⟨l_0, j_0, i_0⟩ ∈ C                                   /* assert C ≠ ∅ */
3    for each ⟨l, j, i⟩ ∈ C with i > 0 do {
4      /* split in front of lower bound */
5      ⟨D, ℋ_[w]⟩ := Find(𝒟, ⟨l, i⟩)
6      let u, v ∈ ℕ : ⟨l, v, u⟩ ∈ D ∧ ⟨l, i, i⟩ ⊑ ⟨l, v, u⟩
7      if i ≠ u then {
8        Split(𝒟, ⟨l, i⟩, i − u)
9      }
10     /* split behind upper bound */
11     ⟨D, ℋ_[w]⟩ := Find(𝒟, ⟨l, j⟩)
12     let u, v ∈ ℕ : ⟨l, v, u⟩ ∈ D ∧ ⟨l, j, j⟩ ⊑ ⟨l, v, u⟩
13     if j ≠ v then {
14       Split(𝒟, ⟨l, j⟩, j − u + 1)
15     }
16   }
17   m := 0
18   p := 0
19   while m ≠ width(C) do {
20     /* split from left to right at inner bounds */
21     for each ⟨l, j, i⟩ ∈ C do {
22       ⟨D, ℋ_[w]⟩ := Find(𝒟, ⟨l, i + m⟩)
23       m_p := width(D)
24       p := p + 1
25     }
26     w := min{ m_p | 0 ≤ p < |C| }
27     for each ⟨l, j, i⟩ ∈ C do {
28       Split(𝒟, ⟨l, i + m⟩, w)
29     }
30     /* combine respective sets of chunks */
31     for each ⟨l, j, i⟩ ∈ C do {
32       Union(𝒟, ⟨l, i + m⟩, ⟨l_0, i_0 + m⟩)
33     }
34     /* add bitwise dependencies of ℬ_[m] */
35     ⟨D, ℋ_[w]⟩ := Find(𝒟, ⟨l_0, i_0 + m⟩)
36     ℋ'_[w] := OR ( ⟨D, ℋ_[w]⟩, ⟨D, ℬ_[w]⟩ )                 /* note: ℬ_[w] ≃ ℬ_[m] */
37     𝒟 := 𝒟 \ { ⟨D, ℋ_[w]⟩ }
38     𝒟 := 𝒟 ∪ { ⟨D, ℋ'_[w]⟩ }
39     /* left jump to next inner bound */
40     m := m + w
41   }
42 }
```

Let $\mathcal{D}'$ denote the "result" of Algorithm 4, i.e. $\mathcal{D}'$ denotes the final state of the bitwise decomposition $\mathcal{D}$ after the modifications made by Algorithm 4. Then $\mathcal{D}'$ is a bitwise decomposition for which Equation (6.23) holds.

Theorem 6.30 (Bitwise Decompositions) *Let* $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ *be a bitwise decomposition. Let* $C \subseteq \mathsf{Chunks}_{\langle n_1,\ldots,n_k \rangle}$ *be a set of chunks and* $\mathcal{B}_{[m]}$ *be a bitwise bitvector function with* $m \in \mathbb{N}_+$ *such that* $\mathcal{B}_{[m]} \circ \lambda_C$ *is well defined. Let*

$$\mathcal{D}' := \mathsf{Slice}(\mathcal{D}, \langle C, \mathcal{B}_{[m]} \rangle)$$

and let E *be a system of bitvector equations and let* e *be a bitvector equation for which satisfying solutions are exactly the zeros of* $\mathcal{B}_{[m]} \circ \lambda_C$*. Then the following holds:*

$$\mathcal{D} \text{ is a bitwise decomposition of } E \quad \Longrightarrow \quad \mathcal{D}' \text{ is a bitwise decomposition of } E \cup \{e\}$$

Proof: Algorithm 4 modifies $\mathcal{D}$ by solely applying Split and Union operations, except for lines 35–38. Theorems 6.25 and 6.26 then imply that Algorithm 4 without lines 35–38 again yields a bitwise decomposition of E.

The modifications of $\mathcal{D}$ which are made in lines 35–38 merely consist of replacing the bitwise bitvector functions belonging to sets containing chunks which are overlapping with chunks of C by their bitwise union with the bitwise data dependencies specified by $\mathcal{B}_{[m]}$. Corollary 6.24 then yields that $\mathcal{D}'$ again is a bitwise decomposition of $E \cup \{e\}$. ■

Chapter 7

Normalizing Systems of Bitvector Equations

Chapter 6 explains a technique which computes a bitwise decomposition of a given system E of bitvector equations. The equations of E are assumed to be of a specific form, allowing that satisfiability of each equation can be described by a *bitwise* bitvector function. In this chapter, the notion of *normalized systems of bitvector equations* is defined. Normalized systems of bitvector equations form a specific subclass of system of bitvector equations which comply with the requirements of the decomposition computation. We show that for each system E of bitvector equations an equivalent normalized system E' can be generated, and we present procedures which compute such an E' by *rewriting* of the bitvector equations of E. The normalized system E' is then used to compute a bitwise decomposition of the initial system E.

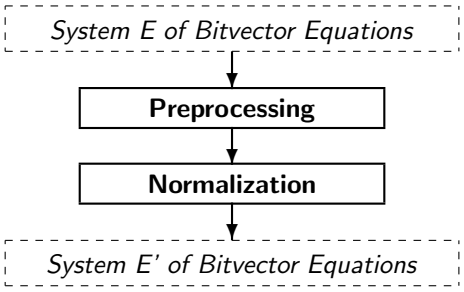


Figure 7.1: Outline

The amount of scaling which can be achieved by the bitvector width reduction technique depends on the widths of the chunks of the computed bitwise decomposition and hence depends on the width of bitwise data flow specified in the normalized system E'. We present a heuristic which optimizes normalization of systems of bitvector equations in this respect by an additional preprocessing of the bitvector equations of E. However, we also show that the general problem of deciding if a bitwise decomposition which is computed from E' in fact is the optimum decomposition of E is an NP-complete problem.

7.1 Characterizing Satisfiability of Bitvector Equations

In Section 6.5, the system E of bitvector equations is assumed to be given in a way such that for each bitvector equation $e \in E$ there exists a bitwise bitvector function $\mathcal{B}^e_{[m]}$ and a set of chunks C_e such that

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \textit{ satisfies } e \quad \Longleftrightarrow \quad \mathcal{B}^e_{[m]} \circ \lambda_{C_e}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = \mathsf{0}_{[m]} \tag{7.1}$$

In fact, the property stated in (7.1) is not a restriction at all and holds for arbitrary bitvector equations $e \in \mathcal{E}(V)$ because it it always possible to characterize satisfiability of any bitvector equation in the Boolean domain.

Theorem 7.1 (Boolean Satisfiability of Bitvector Equations) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables. Let $e \in \mathcal{E}(V)$ be a bitvector equation and let*

$$C \quad := \quad \{\, \langle l, j, i \rangle \in \mathsf{Chunks}_{\langle n_1, \ldots, n_k \rangle} \mid \mathsf{width}(\langle l, j, i \rangle) = 1 \,\}$$

Then there exists a Boolean bitvector function $\mathcal{B}_{[1]}$ of width 1 such that

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(e) \quad \Longleftrightarrow \quad \mathcal{B}_{[1]} \circ \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = \mathsf{0}$$

holds for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$.

Proof: Satisfiability of an arbitrary bitvector equation $e \in \mathcal{E}(V)$ is a finite problem over the m-space domain $\{\mathsf{0}, \mathsf{1}\}^m$ for appropriate $m \in \mathbb{N}_+$. The set of satisfying solutions of e thus can always be coded as a Boolean formula φ involving Boolean variables for all the individual bits of $\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}$. Hence, satisfiability of e can be characterized by a Boolean bitvector function of width 1. ■

The decomposition technique presented in the previous chapter in combination with the strategy outlined in Section 6.5 computes a step-wise refinement of the sets of chunks belonging to the bitwise decomposition of E. However, if Theorem 7.1 is used as a basis to generate the input for the slicing procedure described in Algorithm 4, then the granularities of the bitvector variables which are induced by the resulting bitwise decomposition solely consist of one-bit chunks. All bitwise **BvSAT** problems associated with the sets of chunks are of width 1, and the width reduction technique of Chapter 4 then is of no avail. Therefore, the aim is to describe satisfiability of bitvector equations in the way stated in (7.1), but with sets of chunks and bitwise bitvector functions of largest possible width. In other words, when computing bitwise decompositions, the strategy is to split chunks only if absolutely necessary.

We present a method which for each system E of bitvector equations heuristically computes an equivalent system E' of bitvector equations such that for each bitvector equation $e' \in E'$ a bitwise bitvector function $\mathcal{B}^{e'}_{[m]}$ and a set of chunks $C_{e'}$ exist for which (7.1) holds. E' is

computed by syntactical analysis and by rewriting of the terms of E. Whenever bitwise data dependencies are detected syntactically in E, then corresponding bitwise dependencies exist in E'. For $e' = \langle t'_1, t'_2 \rangle \in E'$ the set $C_{e'}$ is the set of all chunks occurring in t'_1, t'_2 with $\mathsf{width}(C_{e'}) = \mathsf{width}(t'_1) = \mathsf{width}(t'_2)$. For the majority of systems of bitvector equations which up to now have been encountered in practice $m := \mathsf{width}(C_{e'})$ is maximal with respect to E and (7.1).

7.2 Boolean Bitvector Terms and Boolean Bitvector Equations

Let V be a set of bitvector variables. We define the subset $\mathcal{T}_{Bool}(V) \subseteq \mathcal{T}(V)$ of so-called *Boolean bitvector terms*. Boolean bitvector terms are bitvector terms which exclusively contain bitwise Boolean operators and extraction operators. Extraction operators must only take bitvector variables as arguments, but not nested terms. Vice versa, each bitvector variable occurring in a term has to be the argument of a directly following extraction operator defining a specific chunk of the bitvector. Constants are only allowed if they solely consist of either zeros or ones.

Definition 7.2 (Boolean Bitvector Terms) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of k bitvector variables. Then the set $\mathcal{T}_{Bool}(V)$ of **Boolean bitvector terms** over variables of V is defined inductively in the following way:*

- *For each $l \in \{1, \ldots, k\}$ and all $i, j \in \mathbb{N}$ with $0 \leq i \leq j < n_l$, let $\boldsymbol{x}^l_{[n_l]}[j, i] \in \mathcal{T}_{Bool}(V)$.*
- *For each $n \in \mathbb{N}_+$, let $\mathsf{0}_{[n]} \in \mathcal{T}_{Bool}(V)$ and $\mathsf{1}_{[n]} \in \mathcal{T}_{Bool}(V)$.*
- *If $t \in \mathcal{T}_{Bool}(V)$, then let $\mathsf{neg}(t) \in \mathcal{T}_{Bool}(V)$.*
- *For $s, t \in \mathcal{T}_{Bool}(V)$ with $\mathsf{width}(s) = \mathsf{width}(t)$ and $\odot \in \{\mathsf{and}, \mathsf{or}, \mathsf{xor}, \mathsf{nand}, \mathsf{nor}, \mathsf{xnor}\}$ let $(s \odot t) \in \mathcal{T}_{Bool}(V)$.* ■

We define an operator which yields the set of all representations of chunks occurring in a Boolean bitvector term (cf. Definition 6.2).

Definition 7.3 (Chunks Occurring in Boolean Bitvector Terms) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of k bitvector variables and let $t \in \mathcal{T}_{Bool}(V)$. The set $\mathsf{Chunks}(t)$ of chunks occurring in t is inductively defined as follows:*

- *If $t \equiv \boldsymbol{x}^l_{[n_l]}[j, i]$ for $l \in \{1, \ldots, k\}$ and $i, j \in \mathbb{N}$ with $0 \leq i \leq j < n_l$, then let $\mathsf{Chunks}(t) := \{ \langle l, j, i \rangle \}$.*
- *If $t \equiv \mathsf{0}_{[n]}$ or $t \equiv \mathsf{1}_{[n]}$ for $n \in \mathbb{N}_+$, then let $\mathsf{Chunks}(t) := \emptyset$.*
- *If $t \equiv \mathsf{neg}(s)$ for $s \in \mathcal{T}_{Bool}(V)$, then let $\mathsf{Chunks}(t) := \mathsf{Chunks}(s)$.*
- *If $t \equiv t_1 \odot t_2$ for $t_1, t_2 \in \mathcal{T}_{Bool}(V)$ and $\odot \in \{\mathsf{and}, \mathsf{or}, \mathsf{xor}, \mathsf{nand}, \mathsf{nor}, \mathsf{xnor}\}$, then let $\mathsf{Chunks}(t) := \mathsf{Chunks}(t_1) \cup \mathsf{Chunks}(t_2)$.* ■

According to Definitions 7.2 and 7.3, all chunks occurring in a Boolean bitvector term t are of the same width.

Proposition 7.4 (Chunks Occurring in Boolean Bitvector Terms) *Let V be a set of bitvector variables and let $t \in \mathcal{T}_{Bool}(V)$ be a Boolean bitvector term. Then* $\mathsf{Chunks}(t)$ *is a homogeneous set of chunks.*

Proof: Correctness of Proposition 7.4 follows from the definitions of well-formed bitvector terms and Boolean bitvector terms. As $\mathcal{T}_{Bool}(V) \subseteq \mathcal{T}(V)$, all Boolean bitvector terms are well-formed, and bitwise Boolean operators can only take argument terms of the same width. ■

Bitvector equations which consist of Boolean bitvector terms are called *Boolean bitvector equations.*

Definition 7.5 (Boolean Bitvector Equations) *Let V be a finite set of bitvector variables. A bitvector equation $e \in \mathcal{E}(V)$ is called a* ***Boolean bitvector equation*** *if $e = \langle t_1, t_2 \rangle$ for Boolean bitvector terms $t_1 \in \mathcal{T}_{Bool}(V)$ and $t_2 \in \mathcal{T}_{Bool}(V)$.* ■

The set of all Boolean bitvector equations over V is denoted by

$$\mathcal{E}_{Bool}(V) \;:=\; \{\, \langle t_1, t_2 \rangle \in \mathcal{E}(V) \;\mid\; t_1, t_2 \in \mathcal{T}_{Bool}(V) \,\}$$

and the set of chunks occurring in a Boolean bitvector equation e is defined as the union of the sets of chunks occurring in the bitvector terms of e.

Definition 7.6 (Chunks Occurring in Boolean Bitvector Equations) *Let V be a set of bitvector variables and let $e \in \mathcal{E}_{Bool}(V)$ be a Boolean bitvector equation with $e = \langle t_1, t_2 \rangle$ for $t_1 \in \mathcal{T}_{Bool}(V)$ and $t_2 \in \mathcal{T}_{Bool}(V)$. Then we define:*

$$\mathsf{Chunks}(e) \;:=\; \mathsf{Chunks}(t_1) \cup \mathsf{Chunks}(t_2).$$ ■

Then Proposition 7.4 and Definition 7.6 yield:

Proposition 7.7 (Chunks Occurring in Boolean Bitvector Equations) *Let V be a set of bitvector variables and let $e \in \mathcal{E}_{Bool}(V)$ be a Boolean bitvector equation. Then* $\mathsf{Chunks}(e)$ *is a homogeneous set of chunks.*

Boolean bitvector equations have the desired property initially stated at the beginning of this chapter, summarized in (7.1).

Theorem 7.8 (Boolean Bitvector Equations) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables. Let $e \in \mathcal{E}_{Bool}(V)$ be a Boolean bitvector equation. Let $C := \mathsf{Chunks}(e)$ and $m := \mathsf{width}(C)$. Then there exists a bitwise bitvector function $\mathcal{B}_{[m]}$ of arity $|C|$ such that*

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(e) \quad \Longleftrightarrow \quad \mathcal{B}_{[m]} \circ \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) = 0_{[m]}$$

holds for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$.

Proof: Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ and $e = \langle t_1, t_2 \rangle \in \mathcal{E}_{Bool}(V)$. Let $C := \mathsf{Chunks}(e)$ and $m := \mathsf{width}(C)$. Let $p := |C|$ and let $\alpha : \{1, \ldots, p\} \longrightarrow C$ be a bijective enumeration of all chunks of C with $\alpha(1) \ll \alpha(2) \ll \ldots \ll \alpha(p)$. For each $t \in \mathcal{T}_{Bool}(V)$ with $\mathsf{width}(t) = m$ and $\mathsf{Chunks}(t) \subseteq C$ and for all $\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]} \in \mathbb{B}_{[m]}$ let

$$\mathcal{B}^t_{[m]} : \underbrace{\mathbb{B}_{[m]} \times \ldots \times \mathbb{B}_{[m]}}_{p} \longrightarrow \mathbb{B}_{[m]}$$

be inductively defined in the following way:

- If $t \equiv \boldsymbol{x}^l_{[n_l]}[j, i]$, then let $q := \alpha^{-1}(\, \langle l, j, i \rangle\,)$ and define $\mathcal{B}^t_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]}) := \boldsymbol{y}^q_{[m]}$.
- If $t \equiv 0_{[m]}$, then define $\mathcal{B}^t_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]}) := 0_{[m]}$.
- If $t \equiv 1_{[m]}$, then define $\mathcal{B}^t_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]}) := 1_{[m]}$.
- If $t \equiv \mathsf{neg}(s)$ for $s \in \mathcal{T}_{Bool}(V)$, then $\mathcal{B}^t_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]}) := \mathsf{neg}(\, \mathcal{B}^s_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]})\,)$.
- If $t \equiv s_1 \odot s_2$ for $s_1, s_2 \in \mathcal{T}_{Bool}(V)$ and $\odot \in \{\mathsf{and}, \mathsf{or}, \mathsf{xor}, \mathsf{nand}, \mathsf{nor}, \mathsf{xnor}\}$, then $\mathcal{B}^t_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]}) := \mathcal{B}^{s_1}_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]}) \;\odot\; \mathcal{B}^{s_2}_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]})$.

Then for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ we have

$$[\![\, t \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;=\; \mathcal{B}^t_{[m]} \circ \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$$

Now, define $\mathcal{B}_{[m]} : \mathbb{B}_{[m]} \times \ldots \times \mathbb{B}_{[m]} \longrightarrow \mathbb{B}_{[m]}$ by

$$\mathcal{B}_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]}) \;:=\; \mathcal{B}^{t_1}_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]}) \;\mathsf{xor}\; \mathcal{B}^{t_2}_{[m]}(\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]})$$

for all $\boldsymbol{y}^1_{[m]}, \ldots, \boldsymbol{y}^p_{[m]} \in \mathbb{B}_{[m]}$. Then $\mathcal{B}_{[m]}$ is bitwise, and for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ we have:

$$\begin{array}{lll}
 & 0_{[m]} \;=\; \mathcal{B}_{[m]} \circ \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) & \\
\Longleftrightarrow & 0_{[m]} \;=\; \mathcal{B}^{t_1}_{[m]}(\, \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})\,) \;\mathsf{xor}\; \mathcal{B}^{t_2}_{[m]}(\, \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})\,) & \\
\Longleftrightarrow & \mathcal{B}^{t_1}_{[m]}(\, \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})\,) \;=\; \mathcal{B}^{t_2}_{[m]}(\, \lambda_C(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})\,) & \\
\Longleftrightarrow & [\![\, t_1 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \;=\; [\![\, t_2 \,]\!](\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) & \\
\Longleftrightarrow & \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \in \mathcal{L}(e) & \blacksquare
\end{array}$$

7.3 Normalized Systems of Bitvector Equations

A bitvector equation e is said to be *normalized* or *in normalized form* if e is a Boolean bitvector equation, or an if-then-else expression of the form

$$\boldsymbol{x}^{i_1}_{[n]} \;=\; \mathsf{ite}(\boldsymbol{x}^{i_2}_{[m]} = \boldsymbol{x}^{i_3}_{[m]},\, \boldsymbol{x}^{i_4}_{[n]},\, \boldsymbol{x}^{i_5}_{[n]})$$

with $\boldsymbol{x}^{i_1}_{[n]}, \boldsymbol{x}^{i_2}_{[m]}, \boldsymbol{x}^{i_3}_{[m]}, \boldsymbol{x}^{i_4}_{[n]}, \boldsymbol{x}^{i_5}_{[n]} \in V$, or an arithmetic expression of the form

$$\boldsymbol{x}^{i_1}_{[n]} \;=\; \boldsymbol{x}^{i_2}_{[n]} \circledcirc \boldsymbol{x}^{i_3}_{[n]}$$

with $\boldsymbol{x}^{i_1}_{[n]}, \boldsymbol{x}^{i_2}_{[n]}, \boldsymbol{x}^{i_3}_{[n]} \in V$ and $\circledcirc \in \{\oplus, \ominus, \circledast\}$.

Definition 7.9 (Normalized Bitvector Equations) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of k bitvector variables. The set $\mathcal{E}_{Norm}(V)$ of **normalized bitvector equations** over variables of V is defined as follows:*

- *For each $e \in \mathcal{E}_{Bool}(V)$, let $e \in \mathcal{E}_{Norm}(V)$.*
- *For each $i_1, \ldots, i_5 \in \{1, \ldots, k\}$ with $n_{i_1} = n_{i_4} = n_{i_5}$ and $n_{i_2} = n_{i_3}$, let $\langle\, \boldsymbol{x}^{i_1}_{[n_{i_1}]},\, \mathsf{ite}(\boldsymbol{x}^{i_2}_{[n_{i_2}]} = \boldsymbol{x}^{i_3}_{[n_{i_3}]}, \boldsymbol{x}^{i_4}_{[n_{i_4}]}, \boldsymbol{x}^{i_5}_{[n_{i_5}]}) \,\rangle \in \mathcal{E}_{Norm}(V)$.*
- *For each $i_1, i_2, i_3 \in \{1, \ldots, k\}$ with $n_{i_1} = n_{i_2} = n_{i_3}$ and each $\circledcirc \in \{\oplus, \ominus, \circledast\}$, let $\langle\, \boldsymbol{x}^{i_1}_{[n_{i_1}]},\, \boldsymbol{x}^{i_2}_{[n_{i_2}]} \circledcirc \boldsymbol{x}^{i_3}_{[n_{i_3}]} \,\rangle \in \mathcal{E}_{Norm}(V)$.* ■

If $E \subseteq \mathcal{E}_{Norm}(V)$ is a system of normalized bitvector equations, then bitwise and non-bitwise data dependencies between chunks of bitvector variables of V, which are imposed by equations of E, are strictly separated. Bitwise data dependencies are described by Boolean bitvector equations, and non-bitwise data dependencies are described by if-then-else and arithmetic equations.

The following procedure which is shown in Algorithm 5 computes a bitwise decomposition of a system $E \subseteq \mathcal{E}_{Norm}(V)$ of normalized bitvector equations according to the strategy which is outlined in Section 6.5.

Lines 2 – 6 initialize the decomposition data structure (induction base), and lines 7 – 23 successively compute the slicing for each bitvector equation of E. Slicing is based on Theorems 7.1 and 7.8. Chunks occurring in Boolean bitvector equations are *not split* before invoking the Slice procedure. Thus, the complete information on bitwise data flow, as contained in a Boolean bitvector equation, is kept, and chunks are split only if required by Slice. For if-then-else expressions and arithmetic expressions the complete data flow conservatively is decomposed into single bits (in Chapter 8 a technique is presented which prevents single-bit decomposition for if-then-else expressions).

Algorithm 5 Bitwise Decomposition of Normalized Systems of Bitvector Equations

Input: $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$, set of bitvector variables,
$E \subseteq \mathcal{E}_{Norm}(V)$, system of normalized bitvector equations

Output: $\mathcal{D}$, bitwise decomposition of E

```
1  decompose(E, V) {
2    for i = 1 to k do {
3      C_i := { ⟨i, n_i − 1, 0⟩ }
4      let B^i_[n_i] : 𝔹_[n_i] ⟶ 𝔹_[n_i] be defined by x_[n_i] ↦ 0_[n_i] for all x_[n_i] ∈ 𝔹_[n_i]
5    }
6    𝒟 := { ⟨C_1, B^1_[n_1]⟩, ..., ⟨C_k, B^k_[n_k]⟩ }
7    for each e ∈ E do {
8      if e ∈ ℰ_Bool(V) then {
9        C := Chunks(e)
10       let B_[m] be defined according to Theorem 7.8
11       𝒟 := Slice(𝒟, ⟨C, B_[m]⟩)
12     }
13     else if e ≡ x^i1_[n_i1] = ite(x^i2_[n_i2] = x^i3_[n_i3], x^i4_[n_i4], x^i5_[n_i5]) then {
14       C := { ⟨l, j, j⟩ | l ∈ {i_1, ..., i_5} ∧ 0 ≤ j < n_l }
15       let B_[1] be defined according to Theorem 7.1
16       𝒟 := Slice(𝒟, ⟨C, B_[1]⟩)
17     }
18     else if e ≡ x^i1_[n_i1] = x^i2_[n_i2] ◉ x^i3_[n_i3] with ◉ ∈ {⊕, ⊖, ⊛} then {
19       C := { ⟨l, j, j⟩ | l ∈ {i_1, i_2, i_3} ∧ 0 ≤ j < n_l }
20       let B_[1] be defined according to Theorem 7.1
21       𝒟 := Slice(𝒟, ⟨C, B_[1]⟩)
22     }
23   }
24   return 𝒟
25 }
```

In the following section it is shown that each system E of bitvector equations can be rewritten into an equivalent normalized system of bitvector equations.

7.4 Normalization of Bitvector Terms and Bitvector Equations

For each system E of bitvector equations, there exists an equivalent system E' of bitvector equations which is in normalized form and which possibly contains additional auxiliary bitvector variables.

Theorem 7.10 (Normalized Systems of Bitvector Equations) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables. Let $E \subseteq \mathcal{E}(V)$ be a system of bitvector equations. Then there exists a set $V' = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^{k+p}_{[n_{k+p}]}\}$ of bitvector variables with $V \subseteq V'$ and $p \in \mathbb{N}$ and $n_{k+1}, \ldots, n_{k+p} \in \mathbb{N}_+$, and there exists a system $E' \subseteq \mathcal{E}_{Norm}(V')$ of normalized bitvector equations such that*

$$E \text{ satisfiable} \quad \Longleftrightarrow \quad E' \text{ satisfiable}$$

and such that for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ we have

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \text{ satisfies } E \quad \Longrightarrow \quad \exists\, \boldsymbol{x}^{k+1}_{[n_{k+1}]} \in \mathbb{B}_{[n_{k+1}]}, \ldots, \boldsymbol{x}^{k+p}_{[n_{k+p}]} \in \mathbb{B}_{[n_{k+p}]} \;:\; \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^{k+p}_{[n_{k+p}]} \rangle \text{ satisfies } E'$$

and for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^{k+p}_{[n_{k+p}]} \in \mathbb{B}_{[n_{k+p}]}$ we have

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^{k+p}_{[n_{k+p}]} \rangle \text{ satisfies } E' \quad \Longrightarrow \quad \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \text{ satisfies } E$$

i.e. each satisfying solution of E can be embedded in a satisfying solution of E', and each satisfying solution of E' yields a satisfying solution of E when restricted to bitvector variables of V. ■

Such an equivalent normalized system E' can explicitly be computed, for example by successively normalizing all equations of E, as it is done in the main loop of Algorithm 6 given below.

Algorithm 6 Normalization of Systems of Bitvector Equations

```
1  normalize(E, V) {
2    E' := ∅
3    while E ≠ ∅ do {
4      let e ∈ E
5      e' := normalize_equation(e, E, V)
6      E := E \ { e }
7      E' := E' ∪ { e' }
8    }
9    return E'
10 }
```

Bitvector equations are normalized by normalizing both the left hand term and the right hand term separately, as shown in Algorithm 7. If necessary, the initial set of bitvector variables V is dynamically augmented by fresh variables, and the initial system E is augmented by additional auxiliary bitvector equations which then are normalized themselves in subsequent iterations of the main loop of Algorithm 6.

Algorithm 7 Normalization of Bitvector Equations

```
1  normalize_equation( ⟨t1, t2⟩, &E, &V) {
2    t1' := normalize_term(t1, E, V)
3    t2' := normalize_term(t2, E, V)
4    return ⟨t1', t2'⟩
5  }
```

The parameters E and V of Algorithm 7 are of type call-by-reference, indicated by the prefix operator &, allowing E and V to be modified in the calling normalize procedure. The same holds for the parameters of Algorithm 8 which normalizes a bitvector term by computing an equivalent set of normalized bitvector terms.

Algorithm 8 Normalization of Bitvector Terms

```
1  normalize_term(t, &E, &V) {
2    t1 := normalize_constants(t)
3    t2 := normalize_non_Bool_expr(t1, E, V)
4    t3 := normalize_extractions(t2, width(t2) − 1, 0)
5    return t3
6  }
```

Normalization of bitvector terms is done in three successive steps denoted by normalize_constants, normalize_non_Bool_expr and normalize_extractions. These steps are described in detail in the following sections.

7.4.1 Normalization of Bitvector Constants

As a first step, all bitvector constants occurring in a bitvector term t are normalized by replacing the constants by concatenations of successive zeros and ones, as illustrated in the following example:

$$1111100010110000 \quad \mapsto \quad 1_{[5]} \otimes 0_{[3]} \otimes 1_{[1]} \otimes 0_{[1]} \otimes 1_{[2]} \otimes 0_{[4]}$$

Bitvector constants are rewritten as concatenations of alternating $0_{[n]}$'s and $1_{[n]}$'s of largest possible widths. Besides that, term structure, sub-terms and bitvector operators of t are not changed in this step. The rewriting of constants is performed by recursively descending into sub-terms of t as shown in Algorithm 9.

Algorithm 9 Normalization of Bitvector Constants

```
1  normalize_constants(t) {
2    switch(t) {
3      case t ≡ x^i_[n_i] then {
4        return t
5      }
6      case t ≡ 0_[n] then {
7        return t
8      }
9      case t ≡ 1_[n] then {
10       return t
11     }
12     case t ≡ 0...0 1 v_{i-1} v_{i-2} ... v_1 v_0 with v_0, ..., v_{i-1} ∈ 𝔹 then {
                  (j)
13       return 0_[j] ⊗ normalize_constants(v_{i-1} ... v_1 v_0)
14     }
15     case t ≡ 1...1 0 v_{i-1} v_{i-2} ... v_1 v_0 with v_0, ..., v_{i-1} ∈ 𝔹 then {
                  (j)
16       return 1_[j] ⊗ normalize_constants(v_{i-1} ... v_1 v_0)
17     }
18     case t ≡ s[j, i] then {
19       return ( normalize_constants(s) )[j, i]
20     }
21     case t ≡ t_1 ⊗ t_2 then {
22       return normalize_constants(t_1) ⊗ normalize_constants(t_2)
23     }
24     case t ≡ neg(s) then {
25       return neg( normalize_constants(s) )
26     }
27     case t ≡ t_1 ⊙ t_2 and ⊙ ∈ {and, or, xor, nand, nor, xnor} then {
28       return normalize_constants(t_1) ⊙ normalize_constants(t_2)
29     }
30     case t ≡ ite(s_1 = s_2, t_1, t_2) then {
31       v_1 := normalize_constants(s_1); v_2 := normalize_constants(s_2)
32       w_1 := normalize_constants(t_1); w_2 := normalize_constants(t_2)
33       return ite(v_1 = v_2, w_1, w_2)
34     }
35     case t ≡ t_1 ◉ t_2 and ◉ ∈ {⊕, ⊖, ⊛} then {
36       return normalize_constants(t_1) ◉ normalize_constants(t_2)
37     }
38   }
39 }
```

Then for all bitvector terms $t \in \mathcal{T}(V)$ the following holds:

$$[\![t]\!] \quad = \quad [\![\mathsf{normalize\_constants}(t)]\!]$$

i.e. normalization of bitvector constants does not alter the interpreting bitvector function of t.

7.4.2 Normalization of non Boolean Expressions

In a second step, all non Boolean subexpressions of a bitvector term t are replaced by fresh auxiliary bitvector variables of appropriate width. The system of bitvector equations is augmented by additional equations which relate auxiliary variables to normalizations of the corresponding subexpressions. Non Boolean expressions consist of concatenations, if-then-else expressions, and arithmetic operators, other expressions are not changed. Rewriting is done recursively according to the algorithm shown below:

Algorithm 10 Normalization of non Boolean Expressions

```
1  normalize_non_Bool_expr(t, &E, &V) {
2    switch(t) {
3      case t ≡ x_[n] then {
4        return t
5      }
6      case t ≡ v_{n-1} ... v_1 v_0 then {
7        return t
8      }
9      case t ≡ s[j,i] then {
10       return ( normalize_non_Bool_expr(s,E,V) )[j,i]
11     }
12     case t ≡ t_1 ⊗ t_2 then {
13       return normalize_concatenations(t,E,V)
14     }
15     case t ≡ neg(s) then {
16       return neg( normalize_non_Bool_expr(s,E,V) )
17     }
18     case t ≡ t_1 ⊙ t_2 and ⊙ ∈ {and,or,xor,nand,nor,xnor} then {
19       return normalize_non_Bool_expr(t_1,E,V) ⊙ normalize_non_Bool_expr(t_2,E,V)
20     }
21     case t ≡ ite(s_1 = s_2, t_1, t_2) then {
22       return normalize_IfThenElse(t,E,V)
23     }
24     case t ≡ t_1 ◉ t_2 and ◉ ∈ {⊕,⊖,⊛} then {
25       return normalize_arithmetics(t,E,V)
26     }
27   }
28 }
```

Normalization of Concatenations

Equations containing concatenations are rewritten without changing the set of satisfying solutions by using extraction operators instead of concatenation operators. Let $t \in \mathcal{T}(V)$ and let $s_1, s_2 \in \mathcal{T}(V)$. Let $n_1 := \mathsf{width}(s_1)$ and $n_2 := \mathsf{width}(s_2)$ and assume $\mathsf{width}(t) = n_1 + n_2$. Then the following holds:

$$\begin{aligned}\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \;\; \textit{satisfies} \;\; t = s_1 \otimes s_2 \quad \Longleftrightarrow& \\ \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \;\; \textit{satisfies} \;\; t\,[n_1 + n_2 - 1, n_2] = s_1 \quad \wedge& \\ \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \;\; \textit{satisfies} \;\; t\,[n_2 - 1, 0] = s_2&\end{aligned}$$

Thus, concatenations can be eliminated by replacing each sub-term which has a concatenation as top-level operator by a fresh auxiliary bitvector variable. This auxiliary variable is added to the set of bitvector variables V, and the set of bitvector equations E which is to be normalized is augmented by two new equations assigning the upper and lower chunk of the new variable to the respective left hand side and right hand side term of the concatenation operator.

$$\ldots \;=\; \ldots \;\; s_1 \otimes s_2 \;\; \ldots \qquad \mapsto \qquad \left\{ \begin{array}{lcl} \ldots & = & \ldots \;\; A \;\; \ldots \\ A[high] & = & s_1 \\ A[low] & = & s_2 \end{array} \right.$$

Normalization of concatenations is performed by the following algorithm:

Algorithm 11 Normalization of Concatenations

```
1  normalize_concatenations( s1 ⊗ s2, &E, &V ) {
2    k := |V|; m1 := width(s1); m2 := width(s2); m := m1 + m2; V := V ∪ { x^{k+1}_{[m]} }
3    E := E ∪ { x^{k+1}_{[m]}[m − 1, m2] = s1,  x^{k+1}_{[m]}[m2 − 1, 0] = s2 }
4    return x^{k+1}_{[m]}
5  }
```

Note that adding the two new equations to E (which is of type call-by-reference) guarantees that both equations are later on normalized by Algorithm 6.

Normalization of If-Then-Else Expressions

Bitvector equations which are not already in normalized form, but contain if-then-else expressions, are rewritten into a set of equations according to the following scheme:

$$\ldots \;=\; \ldots \;\; \mathsf{ite}(s_1 = s_2, t_1, t_2) \;\; \ldots \qquad \mapsto \qquad \left\{ \begin{array}{lcl} \ldots & = & \ldots \;\; A \;\; \ldots \\ A & = & \mathsf{ite}(B = C, D, E) \\ B & = & s_1 \\ C & = & s_2 \\ D & = & t_1 \\ E & = & t_2 \end{array} \right.$$

The algorithm shown below replaces expressions $\mathsf{ite}(s_1 = s_2, t_1, t_2)$ for which at least one of the subs-terms s_1, s_2, t_1, t_2 is no atomic bitvector variable by a fresh auxiliary bitvector variable. Also, for each non atomic term of s_1, s_2, t_1, t_2, a fresh variable is generated. New bitvector equations assigning the fresh variables to their respective subterms and to the normalized ite expression are added to E.

Algorithm 12 Normalization of If-Then-Else Expressions

```
1  normalize_IfThenElse( ite(s1 = s2, t1, t2), &E, &V ) {
2    if s1 ∈ V then {
3      v1 := s1
4    } else {
5      k := |V|; m := width(s1); V := V ∪ { x_[m]^(k+1) }; E := E ∪ { x_[m]^(k+1) = s1 }; v1 := x_[m]^(k+1)
6    }
7    if s2 ∈ V then {
8      v2 := s2
9    } else {
10     k := |V|; m := width(s2); V := V ∪ { x_[m]^(k+1) }; E := E ∪ { x_[m]^(k+1) = s2 }; v2 := x_[m]^(k+1)
11   }
12   if t1 ∈ V then {
13     w1 := t1
14   } else {
15     k := |V|; m := width(t1); V := V ∪ { x_[m]^(k+1) }; E := E ∪ { x_[m]^(k+1) = t1 }; w1 := x_[m]^(k+1)
16   }
17   if t2 ∈ V then {
18     w2 := t2
19   } else {
20     k := |V|; m := width(t2); V := V ∪ { x_[m]^(k+1) }; E := E ∪ { x_[m]^(k+1) = t2 }; w2 := x_[m]^(k+1)
21   }
22   if {s1, s2, t1, t2} ⊆ V then {
23     return t
24   } else {
25     k := |V|; m := width(t1); V := V ∪ { x_[m]^(k+1) }; E := E ∪ { x_[m]^(k+1) = ite(v1 = v2, w1, w2) }
26     return x_[m]^(k+1)
27   }
28 }
```

Normalization of Arithmetic Expressions

In a similar way, arithmetic expressions which contain at least one non atomic subexpression are normalized by adding auxiliary variables and new equations as illustrated by the following scheme.

$$\ldots \;=\; \ldots\; s_1 \circledcirc s_2 \;\ldots \qquad \mapsto \qquad \left\{ \begin{array}{lcl} \ldots & = & \ldots\; A \;\ldots \\ A & = & B \circledcirc C \\ B & = & s_1 \\ C & = & s_2 \end{array} \right.$$

Algorithm 13 processes arithmetic expressions in the same fashion as Algorithm 12 operates for if-then-else expressions.

Algorithm 13 Normalization of Arithmetic Expressions

```
1  normalize_arithmetics( t1 ◉ t2, &E, &V ) {
2    if t1 ∈ V then {
3      v1 := t1
4    } else {
5      k := |V|; m := width(t1); V := V ∪ { x_[m]^(k+1) }; E := E ∪ { x_[m]^(k+1) = t1 }; v1 := x_[m]^(k+1)
6    }
7    if t2 ∈ V then {
8      v2 := t2
9    } else {
10     k := |V|; m := width(t2); V := V ∪ { x_[m]^(k+1) }; E := E ∪ { x_[m]^(k+1) = t2 }; v2 := x_[m]^(k+1)
11   }
12   if {t1, t2} ⊆ V then {
13     return t
14   } else {
15     k := |V|; m := width(t1); V := V ∪ { x_[m]^(k+1) }; E := E ∪ { x_[m]^(k+1) = v1 ◉ v2 }
16     return x_[m]^(k+1)
17   }
18 }
```

7.4.3 Normalization of Extractions

Extraction expressions are normalized by recursively traversing a bitvector term from top-level operator to leaf-terms. Extraction expressions are rewritten on-the-fly according to the following rules until extractions only occur directly in front of bitvector variables.

$$\begin{array}{lcl} (v_{n-1} \ldots v_1 v_0)[j,i] & \mapsto & v_j \ldots v_{i+1} v_i \\ (\, t_{[n]}[l,k]\,)[j,i] & \mapsto & t_{[n]}[k+j, k+i] \\ (\, s_{[n]} \;\odot\; t_{[n]}\,)[j,i] & \mapsto & s_{[n]}[j,i] \;\odot\; t_{[n]}[j,i] \\ (\, \mathsf{neg}(t_{[n]})\,)[j,i] & \mapsto & \mathsf{neg}(\, t_{[n]}[j,i]\,) \end{array}$$

The following procedure normalize_extractions(t, b, a) analyzes the extraction term $t[b,a]$ and recursively migrates all extraction expressions towards sub-terms.

Algorithm 14 Normalization of Extractions

```
1  normalize_extractions(t, b, a) {
2    switch(t) {
3      case t ≡ x_[n] then {
4        return x_[n][b, a]
5      }
6      case t ≡ v_{n-1} ... v_1 v_0 then {
7        return v_b ... v_a
8      }
9      case t ≡ s[j, i] then {
10       return normalize_extractions(t, i + b, i + a)
11     }
12     case t ≡ t_1 ⊗ t_2 then {
13       m_1 := width(t_1); m_2 := width(t_2)
14       if b < m_2 then {
15         return normalize_extractions(t_2, b, a)
16       } else if a ≥ m_2 then {
17         return normalize_extractions(t_1, b − m_2, a − m_2)
18       } else {
19         return normalize_extractions(t_1, b − m_2, 0) ⊗ normalize_extractions(t_2, m_2 − 1, a)
20       }
21     }
22     case t ≡ neg(s) then {
23       return neg( normalize_extractions(s, b, a) )
24     }
25     case t ≡ t_1 ⊙ t_2 and ⊙ ∈ {and, or, xor, nand, nor, xnor} then {
26       return normalize_extractions(t_1, b, a) ⊙ normalize_extractions(t_2, b, a)
27     }
28     case t ≡ ite(s_1 = s_2, t_1, t_2) then {
29       return t;
30     }
31     case t ≡ t_1 ◉ t_2 and ◉ ∈ {⊕, ⊖, ⊛} then {
32       return t;
33     }
34   }
35 }
```

Note that extractions operating on non Boolean expressions, such as $\mathsf{ite}(s_1 = s_2, t_1, t_2)[b, a]$ and $(t_1 \circledcirc t_2)[b, a]$, are not further treated in lines 28 – 33 of Algorithm 14 because such nested expressions are resolved in Algorithm 10. As a conclusion, the following holds

$$[\![t]\!] \quad = \quad [\![\mathsf{normalize\_extractions}(t, \mathsf{width}(t) - 1, 0)]\!]$$

for all bitvector terms $t \in \mathcal{T}(V)$.

7.5 Preprocessing

The results of the decomposition procedures which have been presented in the prior sections in many cases can greatly be improved by an additional preprocessing of the bitvector equations. Preprocessing of bitvector equations essentially consists of rewriting bitvector terms containing operators which enforce unnecessary slicing. In the following, an overview on some of the most important examples of such cases is given.

7.5.1 Elimination of Concatenations

During the normalization of bitvector equations, Algorithm 11 eliminates concatenations by generating an auxiliary variable $\boldsymbol{x}_{[m]}^{k+1}$ which then is split into two chunks for which two new bitvector equations are generated. This splitting of $\boldsymbol{x}_{[m]}^{k+1}$ induces a splitting of the sets of related chunks of a bitwise decomposition, as seen in Sections 6.5, 6.6 and 6.7. For some concatenation expressions this splitting is unnecessary and can be avoided by rewriting concatenation terms before normalization, as done in the following examples:

Combining Continuous Extractions

$$t[j, a+1] \ \otimes \ t[a, i] \qquad \mapsto \qquad t[j, i]$$

Combining Zeros and Ones in Constants

$$\begin{aligned} w_j \ldots w_0 \underbrace{0 \ldots 0}_{n} \ \otimes \ \underbrace{0 \ldots 0}_{m} v_i \ldots v_0 \qquad &\mapsto \qquad w_j \ldots w_0 \underbrace{0 \ldots 0}_{n+m} v_i \ldots v_0 \\ w_j \ldots w_0 \underbrace{1 \ldots 1}_{n} \ \otimes \ \underbrace{1 \ldots 1}_{m} v_i \ldots v_0 \qquad &\mapsto \qquad w_j \ldots w_0 \underbrace{1 \ldots 1}_{n+m} v_i \ldots v_0 \end{aligned}$$

Optimizing Extractions applied to Concatenations

$$(\, s \otimes t \,)[j, i] \qquad \mapsto \qquad \begin{cases} t[j, i] & \textit{if } \ j < \mathsf{width}(t) \\ s[j - \mathsf{width}(t), i - \mathsf{width}(t)] & \textit{if } \ i \geq \mathsf{width}(t) \\ s[j - \mathsf{width}(t), 0] \ \otimes \ t[\mathsf{width}(t) - 1, i] & \textit{else} \end{cases}$$

7.5.2 Elimination of If-Then-Else Expressions

Algorithm 5 computes a bitwise decomposition by describing the data dependencies caused by if-then-else expressions by a bitvector function of one-bit width according to Theorem 7.1 (see Chapter 8 for an improved treatment of if-then-else expressions). The unfortunate result is a splitting of all dependent chunks into single bits. The following rewriting rules completely eliminate if-then-else expressions in certain situations.

If-Then-Else Expressions with Trivial Conditionals

$$\begin{aligned} \mathsf{ite}(s = s, t_1, t_2) &\quad\mapsto\quad t_1 \\ \mathsf{ite}(s = \mathsf{neg}(s), t_1, t_2) &\quad\mapsto\quad t_2 \end{aligned}$$

If-Then-Else Expressions with Trivial Conclusions

$$\mathsf{ite}(s_1 = s_2, t, t) \quad\mapsto\quad t$$

If-Then-Else Expressions Containing Bitvector Constants in Conditionals

$$\mathsf{ite}(v_{n-1} \ldots v_0 = w_{n-1} \ldots w_0, t_1, t_2) \quad\mapsto\quad \begin{cases} t_1 & \textit{if} \quad v_{n-1} \ldots v_0 = w_{n-1} \ldots w_0 \\ t_2 & \textit{else} \end{cases}$$

Nested If-Then-Else Expressions Containing Bitvector Constants

$$\left.\begin{array}{c} \mathsf{ite}(\, \mathsf{ite}(s_1 = s_2, v_{n-1} \ldots v_0, w_{n-1} \ldots w_0) = v_{n-1} \ldots v_0, t_1, t_2\,) \\ \textit{with} \quad v_{n-1} \ldots v_0 \neq w_{n-1} \ldots w_0 \end{array}\right\} \quad\mapsto\quad \mathsf{ite}(s_1 = s_2, t_1, t_2)$$

$$\left.\begin{array}{c} \mathsf{ite}(\, \mathsf{ite}(s_1 = s_2, v_{n-1} \ldots v_0, w_{n-1} \ldots w_0) = w_{n-1} \ldots w_0, t_1, t_2\,) \\ \textit{with} \quad v_{n-1} \ldots v_0 \neq w_{n-1} \ldots w_0 \end{array}\right\} \quad\mapsto\quad \mathsf{ite}(s_1 = s_2, t_2, t_1)$$

Resolving of If-Then-Else Equations

$$t = \mathsf{ite}(s_1 = s_2, t, \mathsf{neg}(t)) \quad\mapsto\quad s_1 = s_2$$

$$t = \mathsf{ite}(s_1 = s_2, u, \mathsf{neg}(t)) \quad\mapsto\quad \begin{cases} t = u \\ s_1 = s_2 \end{cases}$$

7.5.3 Elimination of Arithmetic Expressions

In Algorithm 5, data dependencies of arithmetic expressions are also characterized by bitvector functions of one-bit width. In the following cases arithmetic expressions can be rewritten (eliminated) in a way such that splitting into one-bit chunks can be avoided.

Trivial Arithmetics

$$\begin{aligned}
\boldsymbol{x}_{[n]} \oplus \mathsf{0}_{[n]} &\mapsto \boldsymbol{x}_{[n]} \\
\boldsymbol{x}_{[n]} \ominus \mathsf{0}_{[n]} &\mapsto \boldsymbol{x}_{[n]} \\
\boldsymbol{x}_{[n]} \circledast \mathsf{0}_{[n]} &\mapsto \mathsf{0}_{[n]} \\
\boldsymbol{x}_{[n]} \circledast \mathsf{1}_{[n]} &\mapsto \boldsymbol{x}_{[n]}
\end{aligned}$$

Expressing Certain Additions and Multiplications by Left-Shifts

$$\begin{aligned}
\boldsymbol{x}_{[n]} \oplus \boldsymbol{x}_{[n]} &\mapsto \boldsymbol{x}_{[n-2,0]} \otimes \mathsf{0}_{[1]} \\
\boldsymbol{x}_{[n]} \circledast \underbrace{\mathsf{0}\ldots\mathsf{0}}_{n-m-1}\,\mathsf{1}\,\underbrace{\mathsf{0}\ldots\mathsf{0}}_{m} &\mapsto \boldsymbol{x}_{[n]}[n-m-1,0] \otimes \mathsf{0}_{[m]}
\end{aligned}$$

7.5.4 Evaluation of Constant Expressions

A further means to simplify bitvector terms and bitvector equations is evaluation of constant expressions for bitwise Boolean operators $\odot$ and arithmetic operators $\circledcirc$, as exemplified below.

$$\begin{aligned}
\mathsf{neg}(v_{n-1}\ldots v_0) &\mapsto [\![\, \mathsf{neg}(\boldsymbol{x}_{[n]}) \,]\!](v_{n-1}\ldots v_0) \\
v_{n-1}\ldots v_0 \;\odot\; w_{n-1}\ldots w_0 &\mapsto [\![\, \boldsymbol{x}_{[n]} \;\odot\; \boldsymbol{y}_{[n]} \,]\!](v_{n-1}\ldots v_0, w_{n-1}\ldots w_0) \\
v_{n-1}\ldots v_0 \;\circledcirc\; w_{n-1}\ldots w_0 &\mapsto [\![\, \boldsymbol{x}_{[n]} \;\circledcirc\; \boldsymbol{y}_{[n]} \,]\!](v_{n-1}\ldots v_0, w_{n-1}\ldots w_0)
\end{aligned}$$

7.6 A Note on Computational Complexity

The width reduction technique presented in this thesis yields the best amount of reduction for bitwise bitvector functions of longest possible width. The longer the widths of the chunks of a bitwise decomposition of a system of bitvector equations are, the better is the amount of reduction that is achieved. Therefore, it is desirable to compute the coarsest possible bitwise decomposition, i.e. a bitwise decomposition which induces the coarsest granularities of the bitvector variables. The coarseness of the granularities depends on the amount of slicing performed during the computation of the bitwise decomposition. The amount of slicing in turn depends on the widths of the Boolean bitvector terms occurring in the normalized system of bitvector equations. In the prior sections we have proposed several normalization heuristics

which produce good results (in terms of coarse granularities) for a large number of test cases. In this section, a closer look at the computational complexity of determining the optimum bitwise decomposition is taken. We show that the general problem of deciding if satisfiability of an arbitrary system of bitvector equations can be described by a bitwise decomposition which contains at least one set of chunks of a given width $m \in \mathbb{N}_+$ is an NP-complete problem. We therefore define the *coarseness* of a bitwise decomposition as follows:

Definition 7.11 (Coarseness of Bitwise Decompositions) *Let $\mathcal{D}$ be a bitwise decomposition with $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ for $q \in \mathbb{N}_+$ and $m_1, \ldots, m_q \in \mathbb{N}_+$. The **coarseness** of $\mathcal{D}$ is defined as*

$$\gamma(\mathcal{D}) \quad := \quad \min \{ m_i \mid 1 \leq i \leq q \}$$

i.e. $\gamma(\mathcal{D})$ yields the minimum width of all chunks of $\mathcal{D}$. ■

Furthermore, we define the following formal decision problem:

Definition 7.12 (Coarsest Bitwise Decomposition) *Let* **CBD** *denote the decision problem whether or not on input $m \in \mathbb{N}_+$ and $E \subseteq \mathcal{E}(V)$ there exists a bitwise decomposition $\mathcal{D}_E$ of E with $\gamma(\mathcal{D}_E) = m$.* ■

Then we can conclude:

Theorem 7.13 (NP-Completeness) **CBD** *is an NP-complete problem.*

Proof: Theorem 7.13 is shown by reducing equivalence of arbitrary Boolean formulae to **CBD**. Therefore, let $k \in \mathbb{N}_+$ and let

$$f : \underbrace{\mathbb{B} \times \ldots \times \mathbb{B}}_{k} \longrightarrow \mathbb{B} \quad \text{and} \quad g : \underbrace{\mathbb{B} \times \ldots \times \mathbb{B}}_{k} \longrightarrow \mathbb{B}$$

be two arbitrary Boolean functions of arity k. Let $V := \{ \boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]} \}$ be a set of bitvector variables of width 2. Then there exist Boolean bitvector terms $t_1, t_2 \in \mathcal{T}_{Bool}(V)$ with

- $\mathsf{width}(t_1) = 1$ and $\mathsf{Chunks}(t_1) = \{ \langle l, 0, 0 \rangle \mid 1 \leq l \leq k \}$
- $\mathsf{width}(t_2) = 1$ and $\mathsf{Chunks}(t_2) = \{ \langle l, 1, 1 \rangle \mid 1 \leq l \leq k \}$
- $[\![t_1]\!](\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) \;=\; f \circ \lambda_{\mathsf{Chunks}(t_1)}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]})$
- $[\![t_2]\!](\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) \;=\; g \circ \lambda_{\mathsf{Chunks}(t_2)}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]})$

for all $\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]} \in \mathbb{B}_{[2]}$.

Let $e \in \mathcal{E}_{Bool}(V)$ be the following bitvector equation

$$t_2 \otimes t_1 \quad = \quad 0_{[2]}$$

and let $E := \{ e \}$. We show that there exists a bitwise decomposition $\mathcal{D}$ of E with $\gamma(\mathcal{D}) = 2$ if and only if $f = g$.

$\Longleftarrow$ Let $f = g$. Define $\mathcal{H}_{[2]} : \mathbb{B}_{[2]} \times \ldots \times \mathbb{B}_{[2]} \longrightarrow \mathbb{B}_{[2]}$ as the k-ary bitwise bitvector function with $\mathcal{H}_{[2]} \simeq f$. Then for all $\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]} \in \mathbb{B}_{[2]}$ the following holds:

$$\begin{array}{ll}
 & \mathcal{H}_{[2]}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) = 0_{[2]} \\
\Longleftrightarrow & \mathcal{H}_{[2]}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]})[1] = 0 \ \wedge \ \mathcal{H}_{[2]}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]})[0] = 0 \\
\Longleftrightarrow & f(\boldsymbol{x}^1_{[2]}[1], \ldots, \boldsymbol{x}^k_{[2]}[1]) = 0 \ \wedge \ f(\boldsymbol{x}^1_{[2]}[0], \ldots, \boldsymbol{x}^k_{[2]}[0]) = 0 \\
\Longleftrightarrow & g(\boldsymbol{x}^1_{[2]}[1], \ldots, \boldsymbol{x}^k_{[2]}[1]) = 0 \ \wedge \ f(\boldsymbol{x}^1_{[2]}[0], \ldots, \boldsymbol{x}^k_{[2]}[0]) = 0 \\
\Longleftrightarrow & g \circ \lambda_{\mathsf{Chunks}(t_2)}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) = 0 \ \wedge \ f \circ \lambda_{\mathsf{Chunks}(t_1)}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) = 0 \\
\Longleftrightarrow & [\![t_2]\!](\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) = 0 \ \wedge \ [\![t_1]\!](\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) = 0 \\
\Longleftrightarrow & [\![t_2]\!](\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) \ \otimes \ [\![t_1]\!](\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) = 0_{[2]} \\
\Longleftrightarrow & [\![t_2 \ \otimes \ t_1]\!](\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) = 0_{[2]} \\
\Longleftrightarrow & \langle \boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]} \rangle \ \textit{satisfies} \ e
\end{array}$$

Let $C := \{ \langle l, 1, 0 \rangle \mid 1 \leq l \leq k \}$. Then $\mathcal{D} := \{ \langle C, \mathcal{H}_{[2]} \rangle \}$ is a bitwise decomposition of e.

$\Longrightarrow$ Let $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ be a bitwise decomposition of E with $\gamma(\mathcal{D}) = 2$. Then $m_i = 2$ for all $i \in \{1, \ldots, q\}$. Let $\mathcal{H}_{[2]} : \underbrace{\mathbb{B}_{[2]} \times \ldots \times \mathbb{B}_{[2]}}_{k} \longrightarrow \mathbb{B}_{[2]}$ be defined by

$$\mathcal{H}_{[2]}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) \quad := \quad \mathcal{B}^1_{[m_1]} \circ \lambda_{C_1}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) \ \ \mathsf{or} \ \ \ldots \ \ \mathsf{or} \ \ \mathcal{B}^q_{[m_q]} \circ \lambda_{C_q}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]})$$

for all $\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]} \in \mathbb{B}_{[2]}$. Then $\mathcal{H}_{[2]}$ is a bitwise bitvector function, because $C_1, \ldots, C_q$ are independent sets of chunks. Let $C := C_1 \cup \ldots \cup C_q$. Then $\{ \langle C, \mathcal{H}_{[2]} \rangle \}$ is a bitwise decomposition of E, and for all $\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]} \in \mathbb{B}_{[2]}$ we have:

$$\langle \boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]} \rangle \ \textit{satisfies} \ E \quad \Longleftrightarrow \quad \mathcal{H}_{[2]}(\boldsymbol{x}^1_{[2]}, \ldots, \boldsymbol{x}^k_{[2]}) = 0_{[2]}$$

Thus the following holds:

$$\begin{array}{l}
g(\boldsymbol{x}^1_{[2]}[1], \ldots, \boldsymbol{x}^k_{[2]}[1]) \ \otimes \ f(\boldsymbol{x}^1_{[2]}[0], \ldots, \boldsymbol{x}^k_{[2]}[0]) = 0_{[2]} \quad \Longleftrightarrow \\
\qquad\qquad \mathcal{H}_{[1]}(\boldsymbol{x}^1_{[2]}[1], \ldots, \boldsymbol{x}^k_{[2]}[1]) \ \otimes \ \mathcal{H}_{[1]}(\boldsymbol{x}^1_{[2]}[0], \ldots, \boldsymbol{x}^k_{[2]}[0]) = 0_{[2]}
\end{array} \tag{7.2}$$

From (7.2) we then conclude

$$g(\boldsymbol{x}^1_{[2]}[1], \ldots, \boldsymbol{x}^k_{[2]}[1]) = 0 \quad \Longleftrightarrow \quad \mathcal{H}_{[1]}(\boldsymbol{x}^1_{[2]}[1], \ldots, \boldsymbol{x}^k_{[2]}[1]) = 0 \tag{7.3}$$

and

$$f(\boldsymbol{x}^1_{[2]}[0],\ldots,\boldsymbol{x}^k_{[2]}[0]) = 0 \quad \Longleftrightarrow \quad \mathcal{H}_{[1]}(\boldsymbol{x}^1_{[2]}[0],\ldots,\boldsymbol{x}^k_{[2]}[0]) = 0 \tag{7.4}$$

Since f, g and $\mathcal{H}_{[1]}$ are Boolean functions of width 1, (7.3) and (7.4) imply:

$$g(\boldsymbol{x}^1_{[2]}[1],\ldots,\boldsymbol{x}^k_{[2]}[1]) = \mathcal{H}_{[1]}(\boldsymbol{x}^1_{[2]}[1],\ldots,\boldsymbol{x}^k_{[2]}[1]) \tag{7.5}$$

and

$$f(\boldsymbol{x}^1_{[2]}[0],\ldots,\boldsymbol{x}^k_{[2]}[0]) = \mathcal{H}_{[1]}(\boldsymbol{x}^1_{[2]}[0],\ldots,\boldsymbol{x}^k_{[2]}[0]) \tag{7.6}$$

As (7.5) and (7.6) hold for all $\boldsymbol{x}^1_{[2]},\ldots,\boldsymbol{x}^k_{[2]} \in \mathbb{B}_{[2]}$, we thus have $f = \mathcal{H}_{[1]}$ and $g = \mathcal{H}_{[1]}$, and consequently $f = g$. ■

The reasoning which is used in the proof of Theorem 7.13 given above is based upon the following equivalence:

$$\mathcal{B}_{[n]}(\boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]}) \;\otimes\; \mathcal{B}_{[m]}(\boldsymbol{y}^1_{[m]},\ldots,\boldsymbol{y}^k_{[m]}) \;=\; \mathcal{B}_{[n+m]}(\boldsymbol{x}^1_{[n]} \otimes \boldsymbol{y}^1_{[m]},\ldots,\boldsymbol{x}^k_{[n]} \otimes \boldsymbol{y}^k_{[m]}) \tag{7.7}$$

which holds for all Boolean bitvector functions $\mathcal{B}_{[1]}$ and all $n, m \in \mathbb{N}_+$ and $\boldsymbol{x}^1_{[n]},\ldots,\boldsymbol{x}^k_{[n]} \in \mathbb{B}_{[n]}$ and $\boldsymbol{y}^1_{[m]},\ldots,\boldsymbol{y}^k_{[m]} \in \mathbb{B}_{[m]}$ (note: $\mathcal{B}_{[1]} \simeq \mathcal{B}_{[n]} \simeq \mathcal{B}_{[m]} \simeq \mathcal{B}_{[n+m]}$). Consider, for example, the following bitvector equation:

$$\boldsymbol{x}^1_{[2]} \;\mathsf{and}\; \boldsymbol{x}^2_{[2]} \;=\; 0_{[2]} \tag{7.8}$$

For (7.8) the decomposition technique which has been presented yields the bitwise decomposition $\mathcal{D} := \{\, \langle C, \mathcal{B}_{[2]}\rangle \,\}$ with $C = \{\langle 1,1,0\rangle, \langle 2,1,0\rangle\}$ and $\mathcal{B}_{[2]}(\boldsymbol{a}_{[2]}, \boldsymbol{b}_{[2]}) := \boldsymbol{a}_{[2]} \;\mathsf{and}\; \boldsymbol{b}_{[2]}$, which is the coarsest bitwise decomposition of (7.8). Now consider the bitvector equation:

$$(\boldsymbol{x}^1_{[2]}[1] \;\mathsf{and}\; \boldsymbol{x}^2_{[2]}[1]) \;\otimes\; \mathsf{neg}(\,\mathsf{neg}(\boldsymbol{x}^1_{[2]}[0]) \;\mathsf{or}\; \mathsf{neg}(\boldsymbol{x}^2_{[2]}[0])\,) \;=\; 0_{[2]} \tag{7.9}$$

$\mathcal{D}$ is also a bitwise decomposition of (7.9) because according to de Morgan's law and according to (7.7) for all $\boldsymbol{a}_{[2]}, \boldsymbol{b}_{[2]} \in \mathbb{B}_{[2]}$ we have

$$\boldsymbol{a}_{[2]} \;\mathsf{and}\; \boldsymbol{b}_{[2]} \;=\; (\boldsymbol{a}_{[2]}[1] \;\mathsf{and}\; \boldsymbol{b}_{[2]}[1]) \;\otimes\; \mathsf{neg}(\,\mathsf{neg}(\boldsymbol{a}_{[2]}[0]) \;\mathsf{or}\; \mathsf{neg}(\boldsymbol{b}_{[2]}[0])\,)$$

However, for the equation shown in (7.9) the presented procedures yield a bitwise decomposition $\mathcal{D} := \{\, \langle C_1, \mathcal{H}_{[1]}\rangle, \langle C_2, \mathcal{H}_{[1]}\rangle \,\}$ with $C_1 = \{\langle 1,0,0\rangle, \langle 2,0,0\rangle\}$, $C_2 = \{\langle 1,1,1\rangle, \langle 2,1,1\rangle\}$, and $\mathcal{H}_{[1]}(\boldsymbol{a}_{[1]}, \boldsymbol{b}_{[1]}) := \boldsymbol{a}_{[1]} \;\mathsf{and}\; \boldsymbol{b}_{[1]}$. The problem lies in determining if two different bitvector terms over neighboring chunks describe corresponding bitwise bitvector functions. As we have seen, this is equivalent to deciding if two Boolean formulae specify the same Boolean function which is an NP-complete problem.

Chapter 8

Handling Dynamic Data Dependencies

The decomposition technique as presented in Chapter 6 produces good results for systems of normalized bitvector equations which contain a high degree of pure uniform data dependencies. This is, for example, the case when a system contains a large number of *Boolean bitvector equations* of broad widths. We have furthermore seen two crucial points of the decomposition technique which cause the resulting granularities of the bitvector variables to consist of single-bit chunks. These crucial points are if-then-else operators and arithmetic operators occurring in the normalized bitvector terms, which screw up uniform data dependencies and bitwise data flow.

In property checking of digital circuits, systems of bitvector equations are used to model hardware designs and to describe formal properties. The proposed reduction technique establishes an automated scaling of data path widths and works well if only bitwise operations are performed on a data path. However, if a data path contains extensive arithmetics, or if a data path contains dynamic data dependencies (which are modeled by if-then-else expressions in the bitvector equations), then usually no good amount of reduction can be achieved.

While a broad variety of circuit designs exists which perform only few (or even none) arithmetic operations on the data path, almost every data path contains dynamic data dependencies. Data packages are routed from point to point according to specific conditions depending on the current state of the control path of a design while execution. In many practical applications the property which is to be checked is independent of the specific bit width of the data packages, but in this case the way in which dynamic data dependencies (i.e. if-then-else expressions) are handled in Algorithm 5 prevents a possible scaling of the data path. The techniques which have been presented up to now demonstrate the basic approach to automated data path scaling, but for real world circuit designs would not lead to the good results which, for example, are presented in Chapter 10.

In this chapter, an extension of the decomposition computation technique is presented which makes the proposed reduction technique successfully applicable in practice. We show how the reduction and decomposition techniques can be elegantly enhanced in a way such that dynamic

data dependencies can also be characterized by *bitwise* bitvector functions and such that no splitting of bitvector chunks at all is required for if-then-else expressions during the decomposition computation. The extension consists of a conceptual separating of data dependencies occurring on the data path from data dependencies occurring on the control path.

8.1 Motivation – Separating Control Flow and Data Flow

Let $m, n \in \mathbb{N}_+$ and let $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}$ be bitvector variables. Consider the following bitvector equation:

$$\boldsymbol{x}_{[n]} \;=\; \mathsf{ite}(\, \boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]},\, \boldsymbol{y}_{[n]},\, \boldsymbol{z}_{[n]}\,) \tag{8.1}$$

Then $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}$ are bitvector variables occurring on the control path of (8.1), and $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}$ belong to the data path. The structural bit-level data dependencies imposed on the individual bits are illustrated in the following figures.

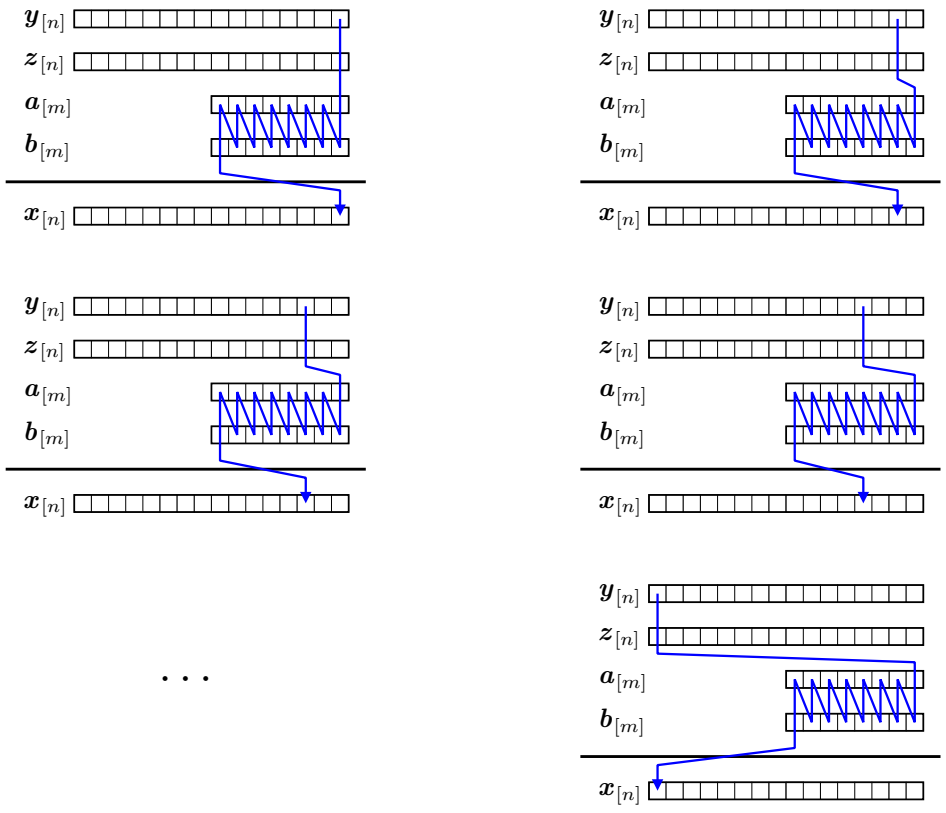


Figure 8.1: Bit-Level Data Dependencies of If-Then-Else Expressions (1)

For each $i \in \{0, \ldots, n-1\}$, the data dependencies of $\boldsymbol{x}_{[n]}[i]$ can informally be described as follows: if $\boldsymbol{a}_{[m]}[0] = \boldsymbol{b}_{[m]}[0]$ and $\boldsymbol{a}_{[m]}[1] = \boldsymbol{b}_{[m]}[1]$ and ... and $\boldsymbol{a}_{[m]}[m-1] = \boldsymbol{b}_{[m]}[m-1]$ then $\boldsymbol{x}_{[n]}[i] = \boldsymbol{y}_{[n]}[i]$, otherwise $\boldsymbol{x}_{[n]}[i] = \boldsymbol{z}_{[n]}[i]$. As can be observed, such bit-level data dependencies are not bitwise.

In Chapters 6 and 7, satisfiability of if-then-else expressions was characterized by bitvector functions of width 1 according to Theorem 7.1. In the case of Equation (8.1), there exists a bitvector function

$$\mathcal{F}_{[1]} : \underbrace{\mathbb{B}_{[1]} \times \ldots \times \mathbb{B}_{[1]}}_{2 \cdot m + 3 \cdot n} \longrightarrow \mathbb{B}_{[1]}$$

of width 1 such that

$$\begin{array}{l} \mathcal{F}_{[1]}(\, \boldsymbol{a}_{[m]}\,[0],\, \ldots\,,\, \boldsymbol{a}_{[m]}\,[\,m-1\,], \\ \qquad \boldsymbol{b}_{[m]}\,[0],\, \ldots\,,\, \boldsymbol{b}_{[m]}\,[\,m-1\,], \\ \qquad \boldsymbol{x}_{[n]}\,[0],\, \ldots\,,\, \boldsymbol{x}_{[n]}\,[\,n-1\,], \\ \qquad \boldsymbol{y}_{[n]}\,[0],\, \ldots\,,\, \boldsymbol{y}_{[n]}\,[\,n-1\,], \\ \qquad \boldsymbol{z}_{[n]}\,[0],\, \ldots\,,\, \boldsymbol{z}_{[n]}\,[\,n-1\,]\,) \;=\; 0_{[1]} \end{array} \qquad \Longleftrightarrow \qquad \boldsymbol{x}_{[n]} = \mathsf{ite}(\boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]})$$

holds for all $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]} \in \mathbb{B}_{[m]}$ and $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]} \in \mathbb{B}_{[n]}$. The bit-level data dependencies of $\mathcal{F}_{[1]}$ imposed by (8.1) are shown below.

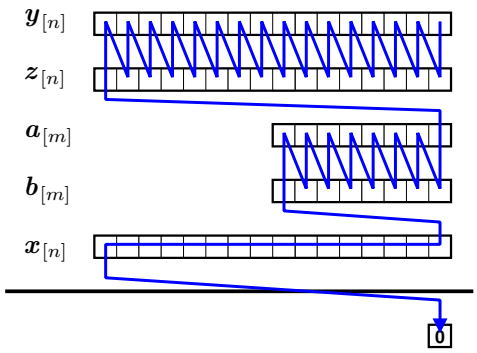


Figure 8.2: Bit-Level Data Dependencies of If-Then-Else Expressions (2)

This straightforward approach causes a complete slicing of $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}$ into single-bit chunks according to lines 13–17 of Algorithm 5. A further effect of the slicing is that all these single-bit chunks are comprised within the same set of chunks of the resulting bitwise decomposition of the system of bitvector equations. The modified decomposition technique which is introduced in this chapter proposes a different treatment of dynamic data dependencies which completely prevents slicing by separating control flow and data flow dependencies.

Each if-then-else expression $\boldsymbol{x}_{[n]} = \mathsf{ite}(\boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]})$ can be rewritten in the following way by introducing an auxiliary bitvector variable $\boldsymbol{d}_{[n]}$:

$$\begin{array}{lcl} \boldsymbol{x}_{[n]} & = & (\ \boldsymbol{y}_{[n]}\ \mathsf{and}\ \mathsf{neg}(\boldsymbol{d}_{[n]})\ )\ \ \mathsf{or}\ \ (\ \boldsymbol{z}_{[n]}\ \mathsf{and}\ \boldsymbol{d}_{[n]}\ ) \\ \boldsymbol{d}_{[n]} & = & \delta_{[n]}(\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}) \end{array} \tag{8.2}$$

where (cf. Definition 3.19)

$$\delta_{[n]}(\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}) \ = \ \begin{cases} 0_{[n]} & \mathsf{if} \quad \boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]} \\ 1_{[n]} & \mathsf{if} \quad \boldsymbol{a}_{[m]} \neq \boldsymbol{b}_{[m]} \end{cases}$$

Then we have

$$(8.1)\ \textit{is satisfiable} \quad \Longleftrightarrow \quad (8.2)\ \textit{is satisfiable}$$

and the following holds:

$$\begin{array}{l} \boldsymbol{x}_{[n]} \ = \ \mathsf{ite}(\boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}) \quad \Longleftrightarrow \\ \qquad\qquad \exists\ \boldsymbol{d}_{[n]} \in \mathbb{B}_{[n]}\ :\ \langle \boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}, \boldsymbol{d}_{[n]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]} \rangle\ \textit{satisfies}\ (8.2) \end{array}$$

The data dependencies which are now imposed by the first equation of (8.2) are shown in Figure 8.3 below.

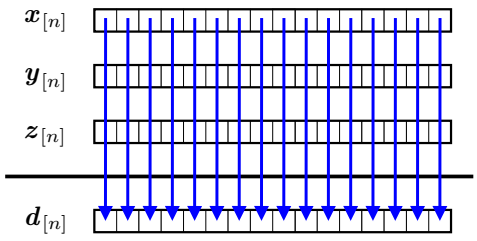


Figure 8.3: Modified Bit-Level Data Dependencies

We observe that this is a completely uniform data flow which describes the data path of (8.1). The control path is described by the second equation of (8.2), and the link between control path and data path is given by $\boldsymbol{d}_{[n]}$. Since in a satisfying solution of (8.2) $\boldsymbol{d}_{[n]}$ can only take two different values, namely $0_{[n]}$ and $1_{[n]}$, Figure 8.3 reveals that in both cases the remaining data dependencies between $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}$ are still bitwise. The reason is that if-then-else expressions basically just resemble a two-cases split, each of which can be characterized by bitvector equalities and disequalities:

$$\boldsymbol{x}_{[n]} = \mathsf{ite}(\boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}) \iff \underbrace{(\boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]} \wedge \boldsymbol{x}_{[n]} = \boldsymbol{y}_{[n]})}_{\text{first case}} \vee \underbrace{(\boldsymbol{a}_{[m]} \neq \boldsymbol{b}_{[m]} \wedge \boldsymbol{x}_{[n]} = \boldsymbol{z}_{[n]})}_{\text{second case}}$$

The means which are necessary to characterize equalities and disequalities within a bitwise framework have already been provided in the previous chapters. We define the following bitvector functions:

$$\begin{aligned}
\mathcal{C}^{=}_{[m]}(\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}) &:= \boldsymbol{a}_{[m]} \mathsf{\ xor\ } \boldsymbol{b}_{[m]} \\
\mathcal{D}^{=}_{[n]}(\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}) &:= \boldsymbol{x}_{[n]} \mathsf{\ xor\ } \boldsymbol{y}_{[n]} \\
\mathcal{C}^{\neq}_{[m]}(\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}) &:= 0_{[m]} \\
\mathcal{D}^{\neq}_{[n]}(\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}) &:= \boldsymbol{x}_{[n]} \mathsf{\ xor\ } \boldsymbol{z}_{[n]}
\end{aligned}$$

Let $I := \{\{1, 2\}\}$. Then $\mathcal{C}^{=}_{[m]}, \mathcal{C}^{\neq}_{[m]}, \mathcal{D}^{=}_{[n]}, \mathcal{D}^{\neq}_{[n]}$ are bitwise bitvector functions for which the following characterization holds:

$$\begin{aligned}
\boldsymbol{x}_{[n]} = \mathsf{ite}(\boldsymbol{a}_{[m]} = \boldsymbol{b}_{[m]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}) \iff & \\
\Big(\underbrace{\langle \boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]} \rangle \textit{ satisfies } \mathbf{BvSAT}(\mathcal{C}^{=}_{[m]}) \wedge \langle \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]} \rangle \textit{ satisfies } \mathbf{BvSAT}(\mathcal{D}^{=}_{[n]})}_{\text{first case}} \Big) & \\
\vee \Big(\underbrace{\langle \boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]} \rangle \textit{ satisfies } \mathbf{BvSAT}(\mathcal{C}^{\neq}_{[m]}, I) \wedge \langle \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]} \rangle \textit{ satisfies } \mathbf{BvSAT}(\mathcal{D}^{\neq}_{[n]})}_{\text{second case}} \Big) &
\end{aligned}$$

To draw a conclusion, we have separated the word-level dependencies occurring on the control path ($\mathcal{C}^{=}_{[m]}$ and $\mathcal{C}^{\neq}_{[m]}$) from those occurring on the data path ($\mathcal{D}^{=}_{[m]}$ and $\mathcal{D}^{\neq}_{[m]}$). We have additionally distinguished between the two cases in which the conditional part of the if-then-else expression evaluates to true and to false, depending on the result of the comparison of $\boldsymbol{a}_{[m]}$ and $\boldsymbol{b}_{[m]}$. Equality of $\boldsymbol{a}_{[m]}$ and $\boldsymbol{b}_{[m]}$ and implied equality of $\boldsymbol{x}_{[n]}$ and $\boldsymbol{y}_{[n]}$ ($\boldsymbol{x}_{[n]}$ and $\boldsymbol{z}_{[n]}$, respectively) is coded in a bitwise fashion within the bitvector functions. Disequality of $\boldsymbol{a}_{[m]}$ and $\boldsymbol{b}_{[m]}$ is coded by an additional disequality constraint I of the respective **BvSAT** problem.

According to Theorem 4.8, satisfiability of pure bitwise **BvSAT** problems can always be reduced to satisfiability of **BvSAT** problems of width 1. Hence, for if-then-else expressions, *any* amount of reduction can be chosen for $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}, \boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}$, as long as $\boldsymbol{x}_{[n]}, \boldsymbol{y}_{[n]}, \boldsymbol{z}_{[n]}$ are reduced to the same width n', and as long as $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}$ are reduced to the same width m', and as long as any disequality constraint for $\boldsymbol{a}_{[m]}, \boldsymbol{b}_{[m]}$ which is imposed by an if-then-else expression is still satisfiable for $\boldsymbol{a}_{[m']}, \boldsymbol{b}_{[m']}$.

8.2 Extending the Decomposition Technique

In this section, the decomposition computation which is presented in Chapter 6 is modified. Lines 13 – 17 of Algorithm 5 are altered in the following fashion:

Algorithm 15 Extended Bitwise Decomposition Computation

```
12      ...
13      else if e ≡ x^{l1}_{[n]} = ite(x^{l2}_{[m]} = x^{l3}_{[m]}, x^{l4}_{[n]}, x^{l5}_{[n]}) then {
14        /* control path */
15        C1 := { ⟨l2, m − 1, 0⟩, ⟨l3, m − 1, 0⟩ }
16        D := Slice(D, ⟨C1, F^2_{0[m]}⟩)
17        /* data path */
18        C2 := { ⟨l1, n − 1, 0⟩, ⟨l4, n − 1, 0⟩, ⟨l5, n − 1, 0⟩ }
19        D := Slice(D, ⟨C2, F^3_{0[n]}⟩)
20      }
21      ...
```

Algorithm 15 causes the sets of chunks of $\mathcal{D}$ which comprise the bits $\boldsymbol{x}^{l_2}_{[m]}[i]$ and $\boldsymbol{x}^{l_3}_{[m]}[i]$ of an if-then-else expression $\boldsymbol{x}^{l_1}_{[n]} = \mathsf{ite}(\boldsymbol{x}^{l_2}_{[m]} = \boldsymbol{x}^{l_3}_{[m]}, \boldsymbol{x}^{l_4}_{[n]}, \boldsymbol{x}^{l_5}_{[n]})$ to be united for all $i \in \{0, \ldots, m-1\}$. The same is done for $\boldsymbol{x}^{l_1}_{[n]}[j]$, $\boldsymbol{x}^{l_4}_{[n]}[j]$ and $\boldsymbol{x}^{l_5}_{[n]}[j]$ for all $j \in \{0, \ldots, n-1\}$. Splitting is only performed in order to be able to unite the respective sets of chunks. Thus, the maximum width of all chunks of $\mathcal{D}$ is preserved with respect to union operations. Note that no additional bitwise data dependencies are imposed on the sets of chunks, as constant zero functions are used as arguments of the Slice procedure.

A bitwise decomposition of E which is computed according to Algorithm 15 does not have the property stated in (6.17) anymore. Let $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$. Although we still have

$$\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \textit{ satisfies } E \implies \forall\, i \in \{1, \ldots, q\} \;:\; \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]})$$

the following, in general, does not hold

$$\forall\, i \in \{1, \ldots, q\} \;:\; \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \implies \langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \textit{ satisfies } E$$

because for if-then-else expressions $\boldsymbol{x}^{l_1}_{[n]} = \mathsf{ite}(\boldsymbol{x}^{l_2}_{[m]} = \boldsymbol{x}^{l_3}_{[m]}, \boldsymbol{x}^{l_4}_{[n]}, \boldsymbol{x}^{l_5}_{[n]})$ occurring in E now chunks of $\boldsymbol{x}^{l_1}_{[n]}, \boldsymbol{x}^{l_4}_{[n]}, \boldsymbol{x}^{l_5}_{[n]}$ are usually separated from chunks of $\boldsymbol{x}^{l_2}_{[m]}, \boldsymbol{x}^{l_3}_{[m]}$, and the data dependencies imposed by the if-then-else operators are not coded in the bitwise bitvector functions of $\mathcal{D}$. In order to be able to describe such dependencies, we define:

Definition 8.1 (If-Then-Else Constraints) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V := \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables and let $E \subseteq \mathcal{E}_{Norm}(V)$ be a normalized system of bitvector equations. Let $\mathsf{ite}(E) \subseteq \{1, \ldots, k\}^5$ denote the set of quintuples for which the following holds:*

$$\langle l_1, l_2, l_3, l_4, l_5 \rangle \in \mathsf{ite}(E) \quad \Longleftrightarrow \quad \langle\, \boldsymbol{x}^{l_1}_{[n_{l_1}]},\ \mathsf{ite}(\boldsymbol{x}^{l_2}_{[n_{l_2}]} = \boldsymbol{x}^{l_3}_{[n_{l_3}]},\ \boldsymbol{x}^{l_4}_{[n_{l_4}]},\ \boldsymbol{x}^{l_5}_{[n_{l_5}]}) \,\rangle \in E$$

*where $l_1, l_2, l_3, l_4, l_5 \in \{1, \ldots, k\}$. Then $\mathsf{ite}(E)$ is called the set of **if-then-else constraints** induced by E.* ■

A satisfying solution $\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle$ of E must satisfy all if-then-else expressions. Thus the following characterization holds:

Corollary 8.2 (Extended Bitwise Decomposition) *Let $k \in \mathbb{N}_+$ and $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V := \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables and let $E \subseteq \mathcal{E}_{Norm}(V)$ be a normalized system of bitvector equations. Let $\mathcal{D} = \{\, \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \,\}$ be a bitwise decomposition of E computed according to Algorithm 15. Then for all $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ the following holds:*

$$\begin{aligned} &\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \text{ satisfies } E \quad \Longleftrightarrow \\ &\qquad \forall\ i \in \{1, \ldots, q\}\ :\ \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \text{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \\ &\quad \wedge\ \forall\ \langle l_1, l_2, l_3, l_4, l_5 \rangle \in \mathsf{ite}(E)\ :\ \boldsymbol{x}^{l_1}_{[n_{l_1}]} = \mathsf{ite}(\boldsymbol{x}^{l_2}_{[n_{l_2}]} = \boldsymbol{x}^{l_3}_{[n_{l_3}]},\ \boldsymbol{x}^{l_4}_{[n_{l_4}]},\ \boldsymbol{x}^{l_5}_{[n_{l_5}]}) \end{aligned}$$ ■

Our goal is to reduce satisfiability of E to satisfiability of a corresponding system E' of bitvector equations over bitvector variables of reduced width (cf. Figure 2.11). In the following chapter, we present in detail how the reduced system E' is generated. We prove that satisfiability of the systems of equations is preserved in a one-to-one fashion, and it is shown how the reduction technique benefits from the extended handling of dynamic data dependencies.

Chapter 9

Applying the Width Reduction Technique

In this chapter, we show how the bitvector width reduction technique which has been presented in Chapter 4 is used in combination with the computation of bitwise decompositions, as presented in Chapters 6 and 8, and in combination with the normalization procedures given in Chapter 7. In Figure 9.1 it is outlined how the three techniques interact in order to simplify the solving of systems of bitvector equations.

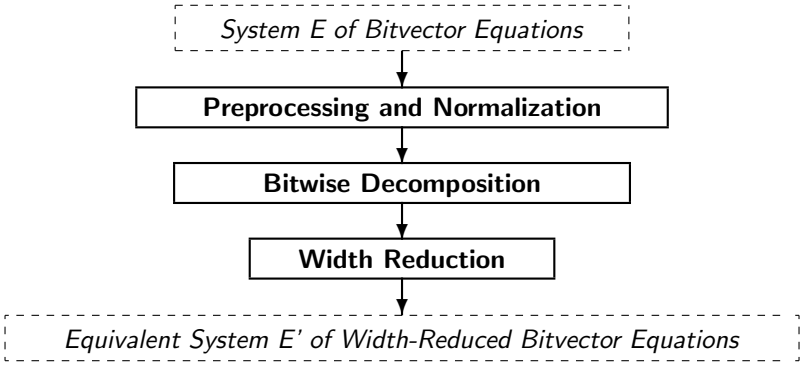


Figure 9.1: Outline

This chapter presents procedures which generate a system E' of bitvector equations over bitvector variables of reduced width from a given normalized system E of bitvector equations. The reduction is based on the extended decomposition technique and the improved handling of dynamic data dependencies which is motivated in Chapter 8. Furthermore, the main reduction theorem for extended bitwise decompositions is presented from which the maximum amount of scaling for each bitvector variable is obtained and which proves correctness of the proposed reduction.

9.1 Generating Equivalent Systems of Bitvector Equations with Reduced Bitvector Widths

In the following, let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V = \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables and let $E \subseteq \mathcal{E}(V)$ be a system of bitvector equations. We have seen that without loss of generality we can assume E to be in normalized form, i.e. $E \subseteq \mathcal{E}_{Norm}(V)$. Let $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ with $q \in \mathbb{N}_+$ and $m_1, \ldots, m_q \in \mathbb{N}_+$ be the bitwise decomposition of E which is computed according to the modified decompose procedure presented in Chapter 8.

The primary objective of the width reduction is to guarantee that the chosen amount of scaling ensures that all if-then-else constraints of E are still satisfiable after scaling. In Example 2.2 and in Chapter 4 it is shown how disequality constraints and the enhanced **BvSAT** problem are used to model such conditions. The set $\mathsf{ite}(E)$ of if-then-else constraints of E induces sets of disequality constraints for the sets of chunks $C_1, \ldots, C_q$ of $\mathcal{D}$ according to the following definition:

Definition 9.1 (Induced Disequality Constraints) *Let $E \subseteq \mathcal{E}_{Norm}(V)$ be a normalized system of bitvector equations, and let $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ be a bitwise decomposition of E computed according to Algorithm 15. Let $I_1, \ldots, I_q$ be sets of disequality constraints such that for all $i \in \{1, \ldots, q\}$ and $a, b \in \mathbb{N}_+$ the following holds:*

$$\{a, b\} \in I_i \iff \begin{array}{l} \text{The bitvector equation } \boldsymbol{x}^{l_1}_{[n]} = \mathsf{ite}(\boldsymbol{x}^{l_2}_{[m]} = \boldsymbol{x}^{l_3}_{[m]}, \boldsymbol{x}^{l_4}_{[n]}, \boldsymbol{x}^{l_5}_{[n]}) \\ \text{occurs in } E, \text{ i.e. } \langle l_1, l_2, l_3, l_4, l_5 \rangle \in \mathsf{ite}(E), \text{ and there exist} \\ c, d \in \{0, \ldots, m-1\} \text{ such that } \langle l_2, d, c \rangle \in C_i \text{ and } \langle l_3, d, c \rangle \in C_i, \\ \text{and } \langle l_2, d, c \rangle \text{ is the } a^{th} \text{ chunk of } C_i \text{ and } \langle l_3, d, c \rangle \text{ is the } b^{th} \text{ chunk} \\ \text{of } C_i \text{ when enumerated according to } \ll. \end{array}$$

*Then $I_1, \ldots, I_q$ are called the sets of **disequality constraints induced by** $\mathcal{D}$ **and** $\mathsf{ite}(E)$.* ■

Let $I_1, \ldots, I_q$ be such sets of disequality constraints. In the following, we will show how to construct a set $V' = \{\boldsymbol{y}^1_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]}\}$ of bitvector variables of reduced widths $N_1, \ldots, N_k \in \mathbb{N}_+$ with $N_1 \leq n_1, \ldots, N_k \leq n_k$ and a system of bitvector equations $E' \subseteq \mathcal{E}_{Norm}(V')$ which is satisfiable if and only if E is satisfiable.

The reduced widths $M_1, \ldots, M_q \in \mathbb{N}_+$ are computed according to the q-fold application of Theorem 4.44, i.e. for each $i \in \{1, \ldots, q\}$ let Theorem 4.44 be instantiated with $I := I_i$ and $n := m_i$ and $\mathcal{F}_{[n]} := \mathcal{B}_{[m_i]}$. The computed reduced width m is then assigned to M_i. Then for all $i \in \{1, \ldots, q\}$ and all $I'_i \subseteq I_i$ we have

$$\textbf{BvSAT}(\mathcal{B}^i_{[m_i]}, I'_i) \text{ is satisfiable} \iff \textbf{BvSAT}(\mathcal{B}^i_{[M_i]}, I'_i) \text{ is satisfiable} \tag{9.1}$$

where $\mathcal{B}^1_{[M_1]} \simeq \mathcal{B}^1_{[m_1]}$, $\ldots$, $\mathcal{B}^q_{[M_q]} \simeq \mathcal{B}^q_{[m_q]}$ denote the corresponding bitwise bitvector functions of widths $M_1, \ldots, M_q$.

Let C be the set of all chunks of $\mathcal{D}$, i.e.

$$C \;:=\; \bigcup_{i \in \{1,\ldots,q\}} C_i$$

and let $G_1, \ldots, G_k \subseteq C$ be the granularities of the bitvector variables $\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}$ which are induced by $\mathcal{D}$, i.e. $G_1 := \mathsf{gran}(\boldsymbol{x}^1_{[n_1]}, \mathcal{D})$, ... , $G_k := \mathsf{gran}(\boldsymbol{x}^k_{[n_k]}, \mathcal{D})$. Then define

$$w : C \;\longrightarrow\; \{M_1, \ldots, M_q\} \qquad \text{with} \qquad w(\alpha) \;:=\; M_i \quad \Longleftrightarrow \quad \alpha \in C_i$$

Furthermore, for each $i \in \{1, \ldots, k\}$ let

$$N_i \;:=\; \sum_{\alpha \in G_i} w(\alpha) \tag{9.2}$$

and define $V' := \{\boldsymbol{y}^1_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]}\}$. The correlation between (chunks of) bitvector variables of V and V' is illustrated in Figure 9.2.

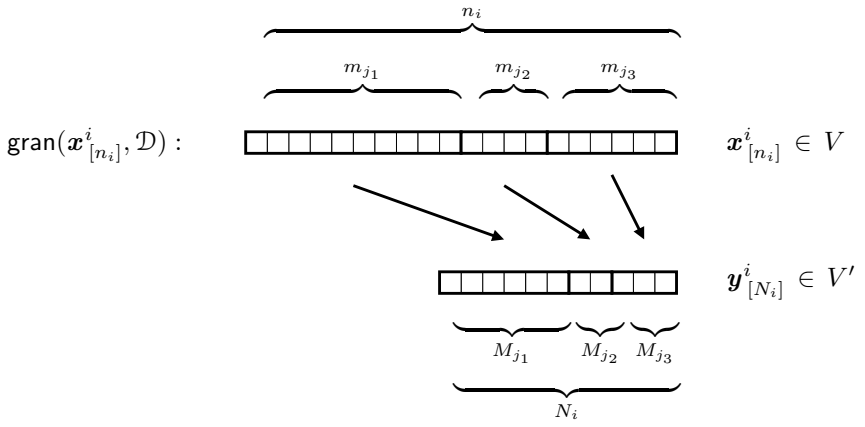


Figure 9.2: Bitvector Width Reduction

Each chunk of the granularity G_i of a bitvector variable $\boldsymbol{x}^i_{[n_i]} \in V$ belongs to a set C_j of $\mathcal{D}$ (and thus is of width m_j) and is related to a specific chunk of $\boldsymbol{y}^i_{[N_i]} \in V'$ (which is of width M_j).

In order to describe this correlation of chunks we define

$$\mathbb{S}_{\mathcal{D}} : \{1, \ldots, k\} \times \mathbb{N} \longrightarrow \mathbb{N}$$

with

$$\langle i, a \rangle \quad \mapsto \quad \sum_{\substack{\alpha \in G_i \\ \alpha \sqsubseteq \langle i, a-1, 0 \rangle}} w(\alpha) \tag{9.3}$$

The function $\mathbb{S}_{\mathcal{D}}$ maps the upper and lower bounds of chunks of bitvector variables of V to the corresponding upper and lower bounds of chunks of bitvector variables of V', as illustrated in Figure 9.3.

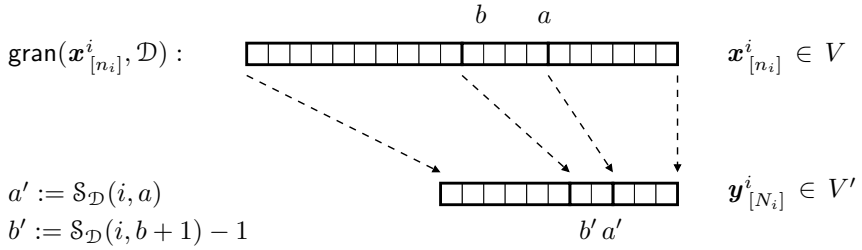


Figure 9.3: Related Upper and Lower Bounds of Chunks

We call $\mathbb{S}_{\mathcal{D}}$ the *reduction function* for $\mathcal{D}$, E and $I_1, \ldots, I_q$. In the following, let $E' \subseteq \mathcal{E}_{Norm}(V')$ be the system of bitvector equations which is constructed from E and $\mathbb{S}_{\mathcal{D}}$ according to Algorithms 16 and 17, which are shown below.

Algorithm 16 Bitvector Width Reduction

```
1  reduce(E, S_D) {
2    E' := ∅
3    for each ⟨t1, t2⟩ ∈ E do {
4      E' := E' ∪ { ⟨ reduce_terms(t1, S_D), reduce_terms(t2, S_D) ⟩ }
5    }
6    return E'
7  }
```

Algorithm 16 generates E' by successively calling Algorithm 17 for all left hand side and right hand side terms of the equations of E. Algorithm 17 replaces all chunk expressions occurring in a normalized bitvector term by chunks expressions of reduced widths according to $\mathbb{S}_{\mathcal{D}}$.

Algorithm 17 Bitvector Width Reduction

```
1  reduce_terms(t, 𝔖_𝒟) {
2    switch(t) {
3      case t ≡ x^i_[n_i][b, a] then {
4        a' := 𝔖_𝒟(i, a);  b' := 𝔖_𝒟(i, b + 1) − 1
5        return y^i_[N_i][b', a']
6      }
7      case t ≡ neg(s) then {
8        return neg( reduce_terms(s, 𝔖_𝒟) )
9      }
10     case t ≡ t_1 ⊙ t_2 and ⊙ ∈ {and, or, xor, nand, nor, xnor} then {
11       return reduce_terms(t_1, 𝔖_𝒟)  ⊙  reduce_terms(t_2, 𝔖_𝒟)
12     }
13     case t ≡ ite(s_1 = s_2, t_1, t_2) then {
14       v_1 := reduce_terms(s_1, 𝔖_𝒟); v_2 := reduce_terms(s_2, 𝔖_𝒟)
15       w_1 := reduce_terms(t_1, 𝔖_𝒟); w_2 := reduce_terms(t_2, 𝔖_𝒟)
16       return ite(v_1 = v_2, w_1, w_2)
17     }
18     case t ≡ t_1 ◉ t_2 and ◉ ∈ {⊕, ⊖, ⊛} then {
19       return reduce_terms(t_1, 𝔖_𝒟)  ◉  reduce_terms(t_2, 𝔖_𝒟)
20     }
21   }
22 }
```

The resulting system E' is syntactically equal to E, except that each chunk of width m_j of a bitvector variable $\boldsymbol{x}^i_{[n_i]}$ of V belonging to a set of chunks C_j, is replaced by the corresponding chunk of $\boldsymbol{y}^i_{[N_i]}$ which is of width M_j. Then the following holds:

Theorem 9.2 (Main Reduction Theorem) *Let $E \subseteq \mathcal{E}_{Norm}(V)$ be a system of normalized bitvector equations. Let $\mathcal{D}$ be an extended bitwise decomposition of E. Let $\mathfrak{S}_{\mathcal{D}}$ be the corresponding reduction function induced by the if-then-else constraints of E. Then the following holds:*

$$E \text{ is satisfiable} \iff \mathsf{reduce}(E, \mathfrak{S}_{\mathcal{D}}) \text{ is satisfiable}$$

where $\mathsf{reduce}(E, \mathfrak{S}_{\mathcal{D}})$ *is the output of Algorithm 16 with input E and $\mathfrak{S}_{\mathcal{D}}$.*

Proof: Let $E' := \mathsf{reduce}(E, \mathfrak{S}_{\mathcal{D}})$. Since $\mathcal{D} = \{ \langle C_1, \mathcal{B}^1_{[m_1]} \rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]} \rangle \}$ is an extended bitwise decomposition of E, according to Corollary 8.2 we have:

$$\begin{aligned} &\langle \boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \rangle \text{ satisfies } E \iff \\ &\qquad \forall\, i \in \{1, \ldots, q\} \;:\; \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \text{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]}) \\ &\quad \wedge\; \forall\, \langle l_1, l_2, l_3, l_4, l_5 \rangle \in \mathsf{ite}(E) \;:\; \boldsymbol{x}^{l_1}_{[n_{l_1}]} = \mathsf{ite}(\boldsymbol{x}^{l_2}_{[n_{l_2}]} = \boldsymbol{x}^{l_3}_{[n_{l_3}]},\, \boldsymbol{x}^{l_4}_{[n_{l_4}]},\, \boldsymbol{x}^{l_5}_{[n_{l_5}]}) \end{aligned} \tag{9.4}$$

Let $f : C \longrightarrow \mathsf{Chunks}_{\langle M_1,\ldots,M_k\rangle}$ be defined by $f(\,\langle l, b, a\rangle\,) := \langle\, l,\, \mathcal{S}_{\mathcal{D}}(l, b+1) - 1,\, \mathcal{S}_{\mathcal{D}}(l, a)\,\rangle$ for all $\langle l, b, a\rangle \in C$. Then $\mathcal{D}' := \{\, \langle f(C_1), \mathcal{B}^1_{[M_1]}\rangle, \ldots, \langle f(C_q), \mathcal{B}^q_{[M_q]}\rangle \,\}$ is an extended bitwise decomposition of E', where $\mathcal{B}^1_{[M_1]} \simeq \mathcal{B}^1_{[m_1]}, \ldots, \mathcal{B}^q_{[M_q]} \simeq \mathcal{B}^q_{[m_q]}$.

Furthermore, we have $\mathsf{ite}(E') = \mathsf{ite}(E)$, and therefore the following holds:

$$\begin{aligned}\langle \boldsymbol{y}^1_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]}\rangle \textit{ satisfies } E' \quad \Longleftrightarrow \\ \forall\, i \in \{1,\ldots,q\} \;:\; \lambda_{f(C_i)}(\boldsymbol{y}^1_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[M_i]}) \\ \wedge\; \forall\, \langle l_1, l_2, l_3, l_4, l_5\rangle \in \mathsf{ite}(E) \;:\; \boldsymbol{y}^{l_1}_{[N_{l_1}]} = \mathsf{ite}(\boldsymbol{y}^{l_2}_{[N_{l_2}]} = \boldsymbol{y}^{l_3}_{[N_{l_3}]},\, \boldsymbol{y}^{l_4}_{[N_{l_4}]},\, \boldsymbol{y}^{l_5}_{[N_{l_5}]}) \end{aligned} \tag{9.5}$$

Correctness of Theorem 9.2 is then obtained from Lemma 9.3. ■

The main reduction theorem characterizes satisfiability of E and E' according to formulae (9.4) and (9.5). Both characterizations are related to each other as stated in the following lemma.

Lemma 9.3 (Bitvector Width Reduction) *Let $k \in \mathbb{N}_+$ and let $n_1, \ldots, n_k \in \mathbb{N}_+$. Let $V := \{\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}\}$ be a set of bitvector variables and let $E \subseteq \mathcal{E}_{Norm}(V)$ be a normalized system of bitvector equations. Let $\mathcal{D} = \{\, \langle C_1, \mathcal{B}^1_{[m_1]}\rangle, \ldots, \langle C_q, \mathcal{B}^q_{[m_q]}\rangle \,\}$ be an extended bitwise decomposition of E computed according to Algorithm 15. Let $I_1, \ldots, I_q$ be the sets of disequality constraints induced by $\mathcal{D}$ and $\mathsf{ite}(E)$. For each $i \in \{1,\ldots,q\}$, let $p_i \in \mathbb{N}_+$ be the number of connected components of the disequality graph $\mathcal{G}_{\neq}(\,|C_i|\,, I_i\,)$, and let $M_i := \min(m_i, \max(1, |C_i| - p_i))$. Let $N_1, \ldots, N_k \in \mathbb{N}_+$ be computed as shown in Equation (9.2), and let f be determined as in the proof of Theorem 9.2. Then the following are equivalent:*

(I) *There exist $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ such that*

- (a) *for all $i \in \{1,\ldots,q\}$ $\lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$ satisfies $\mathbf{BvSAT}(\mathcal{B}^i_{[m_i]})$ and*
- (b) *for all $\langle l_1, l_2, l_3, l_4, l_5\rangle \in \mathsf{ite}(E)$ we have $\boldsymbol{x}^{l_1}_{[n_{l_1}]} = \mathsf{ite}(\boldsymbol{x}^{l_2}_{[n_{l_2}]} = \boldsymbol{x}^{l_3}_{[n_{l_3}]},\, \boldsymbol{x}^{l_4}_{[n_{l_4}]},\, \boldsymbol{x}^{l_5}_{[n_{l_5}]})$.*

(II) *There exist $\boldsymbol{y}^1_{[N_1]} \in \mathbb{B}_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]} \in \mathbb{B}_{[N_k]}$ such that*

- (a) *for all $i \in \{1,\ldots,q\}$ $\lambda_{f(C_i)}(\boldsymbol{y}^1_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]})$ satisfies $\mathbf{BvSAT}(\mathcal{B}^i_{[M_i]})$ and*
- (b) *for all $\langle l_1, l_2, l_3, l_4, l_5\rangle \in \mathsf{ite}(E)$ we have $\boldsymbol{y}^{l_1}_{[N_{l_1}]} = \mathsf{ite}(\boldsymbol{y}^{l_2}_{[N_{l_2}]} = \boldsymbol{y}^{l_3}_{[N_{l_3}]},\, \boldsymbol{y}^{l_4}_{[N_{l_4}]},\, \boldsymbol{y}^{l_5}_{[N_{l_5}]})$.*

Proof: For each $i \in \{1,\ldots,q\}$ let $k_i := |C_i|$.

(I) $\Longrightarrow$ (II): Assume there exist $\boldsymbol{x}^1_{[n_1]} \in \mathbb{B}_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]} \in \mathbb{B}_{[n_k]}$ such that for all $i \in \{1,\ldots,q\}$ $\lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]})$ satisfies $\mathbf{BvSAT}(\mathcal{B}^i_{[m_i]})$ and such that for all $\langle l_1, l_2, l_3, l_4, l_5\rangle \in \mathsf{ite}(E)$ we have $\boldsymbol{x}^{l_1}_{[n_{l_1}]} = \mathsf{ite}(\boldsymbol{x}^{l_2}_{[n_{l_2}]} = \boldsymbol{x}^{l_3}_{[n_{l_3}]},\, \boldsymbol{x}^{l_4}_{[n_{l_4}]},\, \boldsymbol{x}^{l_5}_{[n_{l_5}]})$.

For each $i \in \{1, \ldots, q\}$ let $\boldsymbol{x}^{i,1}_{[m_i]}, \ldots, \boldsymbol{x}^{i,k_i}_{[m_i]} \in \mathbb{B}_{[m_i]}$ such that

$$\langle \boldsymbol{x}^{i,1}_{[m_i]}, \ldots, \boldsymbol{x}^{i,k_i}_{[m_i]} \rangle \;=\; \lambda_{C_i}(\boldsymbol{x}^{1}_{[n_1]}, \ldots, \boldsymbol{x}^{k}_{[n_k]})$$

and let

$$I_i^{\neq} := \{\, \{a,b\} \in I_i \mid \boldsymbol{x}^{i,a}_{[m_i]} \neq \boldsymbol{x}^{i,b}_{[m_i]} \,\}$$

and

$$I_i^{=} := \{\, \{a,b\} \mid a,b \in \{1,\ldots,k_i\} \;\wedge\; \boldsymbol{x}^{i,a}_{[m_i]} = \boldsymbol{x}^{i,b}_{[m_i]} \,\}$$

Then for all $\boldsymbol{z}^{1}_{[m_i]}, \ldots, \boldsymbol{z}^{k_i}_{[m_i]} \in \mathbb{B}_{[m_i]}$ let $\mathcal{G}^{i}_{[m_i]} : \mathbb{B}_{[m_i]} \times \ldots \times \mathbb{B}_{[m_i]} \to \mathbb{B}_{[m_i]}$ be defined by

$$\mathcal{G}^{i}_{[m_i]}(\boldsymbol{z}^{1}_{[m_i]}, \ldots, \boldsymbol{z}^{k_i}_{[m_i]}) \;:=\; \mathcal{B}^{i}_{[m_i]}(\boldsymbol{z}^{1}_{[m_i]}, \ldots, \boldsymbol{z}^{k_i}_{[m_i]}) \quad \mathsf{or} \quad \left(\bigvee_{\{a,b\} \in I_i^{=}} (\boldsymbol{z}^{a}_{[m_i]} \;\mathsf{xor}\; \boldsymbol{z}^{b}_{[m_i]}) \right)$$

Then $\mathcal{G}^{i}_{[m_i]}$ is a bitwise bitvector function, and due to the construction then $\langle \boldsymbol{x}^{i,1}_{[m_i]}, \ldots, \boldsymbol{x}^{i,k_i}_{[m_i]} \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{G}^{i}_{[m_i]}, I_i^{\neq})$. Since $I_i^{\neq} \subseteq I_i$, then Theorem 4.44 yields that $\mathbf{BvSAT}(\mathcal{G}^{i}_{[M_i]}, I_i^{\neq})$ is also satisfiable. Furthermore, according to the definition of corresponding bitwise bitvector functions (see Definitions 3.25 and 3.28), for all $\boldsymbol{z}^{1}_{[M_i]}, \ldots, \boldsymbol{z}^{k_i}_{[M_i]} \in \mathbb{B}_{[M_i]}$ the following holds:

$$\mathcal{G}^{i}_{[M_i]}(\boldsymbol{z}^{1}_{[M_i]}, \ldots, \boldsymbol{z}^{k_i}_{[M_i]}) \;:=\; \mathcal{B}^{i}_{[M_i]}(\boldsymbol{z}^{1}_{[M_i]}, \ldots, \boldsymbol{z}^{k_i}_{[M_i]}) \quad \mathsf{or} \quad \left(\bigvee_{\{a,b\} \in I_i^{=}} (\boldsymbol{z}^{a}_{[M_i]} \;\mathsf{xor}\; \boldsymbol{z}^{b}_{[M_i]}) \right)$$

For each $i \in \{1, \ldots, q\}$ let $\boldsymbol{y}^{i,1}_{[M_i]}, \ldots, \boldsymbol{y}^{i,k_i}_{[M_i]} \in \mathbb{B}_{[M_i]}$ such that $\langle \boldsymbol{y}^{i,1}_{[M_i]}, \ldots, \boldsymbol{y}^{i,k_i}_{[M_i]} \rangle$ is a satisfying solution of $\mathbf{BvSAT}(\mathcal{G}^{i}_{[M_i]}, I_i^{\neq})$. Let $\boldsymbol{y}^{1}_{[N_1]} \in \mathbb{B}_{[N_1]}, \ldots, \boldsymbol{y}^{k}_{[N_k]} \in \mathbb{B}_{[N_k]}$ such that for all $i \in \{1, \ldots, q\}$ we have:

$$\langle \boldsymbol{y}^{i,1}_{[M_i]}, \ldots, \boldsymbol{y}^{i,k_i}_{[M_i]} \rangle \;=\; \lambda_{f(C_i)}(\boldsymbol{y}^{1}_{[N_1]}, \ldots, \boldsymbol{y}^{k}_{[N_k]})$$

Note that such $\boldsymbol{y}^{1}_{[N_1]}, \ldots, \boldsymbol{y}^{k}_{[N_k]}$ exist because $C_1, \ldots, C_q$ and thus $f(C_1), \ldots, f(C_q)$ are *independent* sets of chunks (cf. Chapter 6). Then we can conclude the following:

$$\forall\; i \in \{1, \ldots, q\} \;:\; \lambda_{f(C_i)}(\boldsymbol{y}^{1}_{[N_1]}, \ldots, \boldsymbol{y}^{k}_{[N_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}^{i}_{[M_i]})$$

Now, let $\langle l_1, l_2, l_3, l_4, l_5 \rangle \in \mathsf{ite}(E)$. Then $\boldsymbol{x}^{l_1}_{[n_{l_1}]} = \mathsf{ite}(\boldsymbol{x}^{l_2}_{[n_{l_2}]} = \boldsymbol{x}^{l_3}_{[n_{l_3}]}, \boldsymbol{x}^{l_4}_{[n_{l_4}]}, \boldsymbol{x}^{l_5}_{[n_{l_5}]})$, and either $\boldsymbol{x}^{l_2}_{[n_{l_2}]} \neq \boldsymbol{x}^{l_3}_{[n_{l_3}]}$ and $\boldsymbol{x}^{l_1}_{[n_{l_1}]} = \boldsymbol{x}^{l_5}_{[n_{l_5}]}$, or $\boldsymbol{x}^{l_2}_{[n_{l_2}]} = \boldsymbol{x}^{l_3}_{[n_{l_3}]}$ and $\boldsymbol{x}^{l_1}_{[n_{l_1}]} = \boldsymbol{x}^{l_4}_{[n_{l_4}]}$. According to the construction of $I_1^{\neq}, \ldots, I_q^{\neq}$ and $I_1^{=}, \ldots, I_q^{=}$, in both cases the respective equalities and disequalities

for all chunks of $\boldsymbol{x}^{l_1}_{[n_{l_1}]}, \boldsymbol{x}^{l_2}_{[n_{l_2}]}, \boldsymbol{x}^{l_3}_{[n_{l_3}]}, \boldsymbol{x}^{l_4}_{[n_{l_4}]}, \boldsymbol{x}^{l_5}_{[n_{l_5}]}$ are coded in the $\mathbf{BvSAT}(\mathcal{G}^i_{[M_i]}, I_i^{\neq})$ problems and thus are preserved while reduction, i.e. we have

$$\boldsymbol{x}^{l_2}_{[n_{l_2}]} = \boldsymbol{x}^{l_3}_{[n_{l_3}]} \iff \boldsymbol{y}^{l_2}_{[N_{l_2}]} = \boldsymbol{y}^{l_3}_{[N_{l_3}]}$$

and

$$\boldsymbol{x}^{l_1}_{[n_{l_1}]} = \boldsymbol{x}^{l_4}_{[n_{l_4}]} \implies \boldsymbol{y}^{l_1}_{[N_{l_1}]} = \boldsymbol{y}^{l_4}_{[N_{l_4}]}$$

as well as

$$\boldsymbol{x}^{l_1}_{[n_{l_1}]} = \boldsymbol{x}^{l_5}_{[n_{l_5}]} \implies \boldsymbol{y}^{l_1}_{[N_{l_1}]} = \boldsymbol{y}^{l_5}_{[N_{l_5}]}$$

Hence, we have $\boldsymbol{y}^{l_1}_{[N_{l_1}]} = \mathsf{ite}(\boldsymbol{y}^{l_2}_{[N_{l_2}]} = \boldsymbol{y}^{l_3}_{[N_{l_3}]}, \boldsymbol{y}^{l_4}_{[N_{l_4}]}, \boldsymbol{y}^{l_5}_{[N_{l_5}]})$.

(II) $\Longleftarrow$ (I): Let $\boldsymbol{y}^1_{[N_1]} \in \mathbb{B}_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]} \in \mathbb{B}_{[N_k]}$ such that $\lambda_{f(C_i)}(\boldsymbol{y}^1_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]})$ satisfies $\mathbf{BvSAT}(\mathcal{B}^i_{[M_i]})$ for all $i \in \{1, \ldots, q\}$, and such that for all $\langle l_1, l_2, l_3, l_4, l_5\rangle \in \mathsf{ite}(E)$ we have $\boldsymbol{y}^{l_1}_{[N_{l_1}]} = \mathsf{ite}(\boldsymbol{y}^{l_2}_{[N_{l_2}]} = \boldsymbol{y}^{l_3}_{[N_{l_3}]}, \boldsymbol{y}^{l_4}_{[N_{l_4}]}, \boldsymbol{y}^{l_5}_{[N_{l_5}]})$.

Let $G'_1, \ldots, G'_k \subseteq \mathsf{Chunks}_{\langle N_1, \ldots, N_k\rangle}$ be the granularities of $\boldsymbol{y}^1_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]}$ which are induced by $\mathcal{D}'$ as defined in the proof of Theorem 9.2.

For each $i \in \{1, \ldots, k\}$, let $\alpha_i : \{1, \ldots, |G'_i|\} \longrightarrow G'_i$ be a bijective enumeration of the chunks of G'_i with $\alpha_i(1) \ll \ldots \ll \alpha_i(|G'_i|)$, and let $\beta_i : \{1, \ldots, |G_i|\} \longrightarrow G_i$ be a bijective enumeration of the chunks of G_i with $\beta_i(1) \ll \ldots \ll \beta_i(|G_i|)$.

For each $j \in \{1, \ldots, |G_i|\}$, let $m_{i,j} := \mathsf{width}(\beta_i(j))$, and let $\boldsymbol{x}^i_{[n_i]} \in \mathbb{B}_{[n_i]}$ be defined by $\boldsymbol{x}^i_{[n_i]} :=$

$$\mathsf{signExt}_{[m_{i,|G_i|}]}(\,\lambda_{\{\alpha_i(|G'_i|)\}}(\boldsymbol{y}^1_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]})\,) \otimes \ldots \otimes \mathsf{signExt}_{[m_{i,1}]}(\,\lambda_{\{\alpha_i(1)\}}(\boldsymbol{y}^1_{[N_1]}, \ldots, \boldsymbol{y}^k_{[N_k]})\,)$$

Then the following holds:

$$\forall\, i \in \{1, \ldots, q\} \;:\; \lambda_{C_i}(\boldsymbol{x}^1_{[n_1]}, \ldots, \boldsymbol{x}^k_{[n_k]}) \textit{ satisfies } \mathbf{BvSAT}(\mathcal{B}^i_{[m_i]})$$

Now, let $\langle l_1, l_2, l_3, l_4, l_5\rangle \in \mathsf{ite}(E)$. Then $\boldsymbol{y}^{l_1}_{[N_{l_1}]} = \mathsf{ite}(\boldsymbol{y}^{l_2}_{[N_{l_2}]} = \boldsymbol{y}^{l_3}_{[N_{l_3}]}, \boldsymbol{y}^{l_4}_{[N_{l_4}]}, \boldsymbol{y}^{l_5}_{[N_{l_5}]})$. Equality and disequality of bitvectors is preserved by signed extension. For each $j \in \{1, \ldots, 5\}$ the bitvector $\boldsymbol{x}^{l_j}_{[n_{l_j}]}$ consists of a concatenation of chunks of $\boldsymbol{y}^{l_j}_{[N_{l_j}]}$. In particular, $\boldsymbol{x}^{l_j}_{[n_{l_j}]}$ is constructed such that for all $a, b, c, d \in \mathbb{N}$ the following hold:

$$\begin{array}{lcl}
\boldsymbol{x}^{l_2}_{[n_{l_2}]}[b,a] = \mathsf{signExt}_{[b-a+1]}(\boldsymbol{y}^{l_2}_{[N_{l_2}]}[d,c]) & \iff & \boldsymbol{x}^{l_3}_{[n_{l_3}]}[b,a] = \mathsf{signExt}_{[b-a+1]}(\boldsymbol{y}^{l_3}_{[N_{l_3}]}[d,c]) \\
\boldsymbol{x}^{l_1}_{[n_{l_1}]}[b,a] = \mathsf{signExt}_{[b-a+1]}(\boldsymbol{y}^{l_1}_{[N_{l_1}]}[d,c]) & \iff & \boldsymbol{x}^{l_4}_{[n_{l_4}]}[b,a] = \mathsf{signExt}_{[b-a+1]}(\boldsymbol{y}^{l_4}_{[N_{l_4}]}[d,c]) \\
\boldsymbol{x}^{l_1}_{[n_{l_1}]}[b,a] = \mathsf{signExt}_{[b-a+1]}(\boldsymbol{y}^{l_1}_{[N_{l_1}]}[d,c]) & \iff & \boldsymbol{x}^{l_5}_{[n_{l_5}]}[b,a] = \mathsf{signExt}_{[b-a+1]}(\boldsymbol{y}^{l_5}_{[N_{l_5}]}[d,c])
\end{array}$$

Then we can conclude $\boldsymbol{x}^{l_1}_{[n_{l_1}]} = \mathsf{ite}(\boldsymbol{x}^{l_2}_{[n_{l_2}]} = \boldsymbol{x}^{l_3}_{[n_{l_3}]}, \boldsymbol{x}^{l_4}_{[n_{l_4}]}, \boldsymbol{x}^{l_5}_{[n_{l_5}]})$. ■

Note that the second part of the proof of Lemma 9.3 also reveals how a satisfying solution of the original system E can be obtained from a satisfying solution of the reduced system E'.

9.2 A Note on Order Constraints

In the previous sections of this chapter and in Chapters 5, 6, 7 and 8, only if-then-else expressions containing equality comparisons have been considered. The reason for this was to avoid unnecessary complication and redundant notation. The results which have been presented all remain valid if (some or all) expressions

$$\mathsf{ite}(s_1 = s_2,\, t_1,\, t_2)$$

with bitvector terms s_1, s_2, t_1, t_2 are replaced by

$$\mathsf{ite}(s_1 < s_2,\, t_1,\, t_2)$$

Section 4.9 reveals that comparisons for equality and for strict ordering can be treated completely alike, i.e. the same disequality constraints for the **BvSAT** problems are generated from both types of comparison. Theorem 4.48 yields that the proposed abstraction and the reduced bitvector widths which are computed are sufficient to guarantee existence of satisfying solutions which strictly preserve all order constraints and disequalities of the bitvector values.

Chapter 10

Experimental Results

Reduction of data path widths is a classical approach to cope with design sizes for verification and simulation purposes. It is common practice among chip designers that they manually replace a data path width in the HDL code by a smaller one if verification or simulation of a design becomes too complex. The smaller width is usually simply guessed or chosen by intuition, however this is done without having any guarantee that functional behavior of the scaled design truly corresponds to the behavior of the original design in a one-to-one fashion. Furthermore, it is usually unclear how values of the scaled signals should be interpreted in order to find corresponding values of original size for the initial design.

In the previous chapters, we have shown how the proposed abstraction technique solves these problems and can be used for an automated reduction of data path widths for Bounded Model Checking of digital hardware.

The proposed method has been implemented in C++ in a prototype tool called BooSTER (Boolean String Length Reduction, see [Joh01a]), which now is part of the *Circuit Verification Environment* (CVE, cf. [BS01, Joh01b]) of Infineon Technologies. The tool was tested in several case studies at the Design Automation Department of Siemens Corporation in Munich, Germany, and at the Computer Network Peripherals Department of Infineon Technologies in San Jose, California, in order to examine what amount of scaling can be achieved for real world chip designs, and what types of designs and properties are promising candidates for data path width reduction. The prototype operated as a preprocessor to the property checking tools used at Siemens and Infineon, and two major case studies, in which property checking runtimes on the scaled models of the circuit designs are compared to those achieved on the original designs without preprocessing, are presented in the following sections.

10.1 Central Buffer Address Management Unit

In this section, results are presented which have been achieved on the address management unit of the central buffer of an asynchronous transfer mode (ATM) based switching element, which has been developed at Siemens ICN in Munich, Germany, and which is used in industrial telecommunication chips.

The HDL design consists of roughly 3.000 lines of Verilog source code, which result in a netlist synthesis consisting of approximately 24.000 gates of control and decoding logic, and RAMs with 35.000 memory element cells, which was far too large for the BDD based model checker, so verification took place in a Bounded-Model-Checking approach using the Siemens property checker. Because of its considerable size, and because the design represents a typical class of interface modules, it was chosen as a test case for the data path scaling technique.

The address management unit, in the following called *Q-Unit*, incorporates 16 FIFO queue buffers and complex control logic. It is implemented by a straightforward pipeline operation with almost no data dependent interaction of the pipelined operations. The design has 33 input channels to receive ATM cells.

Data packages are fed on the input channels to the management unit and stored in the FIFOs. Upon request the packages are output on one of 17 output channels again. Many of these cells are empty in which case they are not further processed. Otherwise, they are parallelized and offered to the main circuitry. The main circuitry performs either a write or a read in each clock cycle, following a fixed scan scheme. During a write a cell is polled from an input channel, and its routing information is examined in order to find out on which (possibly multiple) output(s) it shall leave the switching element. Then the cell gets stored in the central buffer, a RAM of several hundreds cells capacity. The address for this storage operation is to be provided by the Q-Unit. A read provides cells to exactly one output channel that is specified by the scan scheme. To serve it, the Q-Unit must deliver the central buffer address of the cell that is to be output over the requested output channel. A block diagram is shown in Figure 10.1.

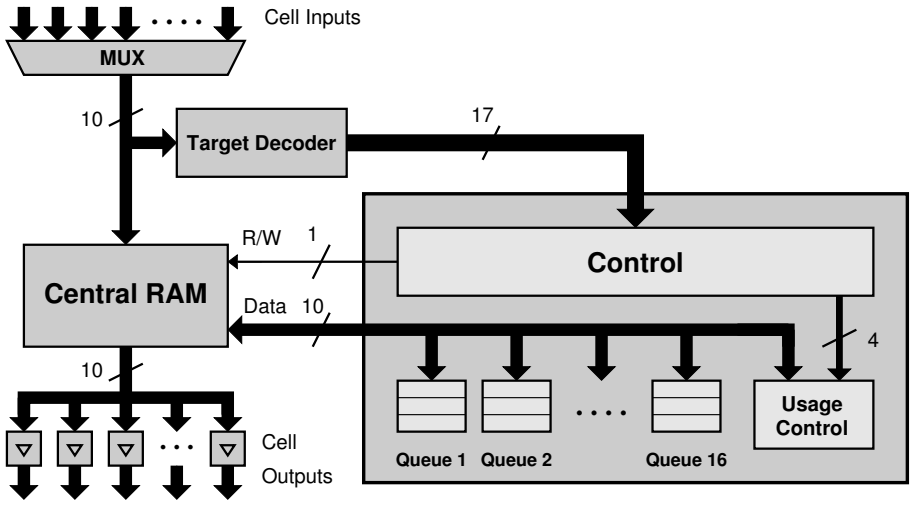


Figure 10.1: RTL Block Diagram of the Q-Unit

Each queue can receive a read, a write, or no request. The normal operation can be disturbed by buffer overflows where a cell should be input but no unused address is available in the central buffer, and by queue overflows where a maximum number of cells that wait to be output over one dedicated output channel would be exceeded. The Q-Unit must support requests such that pipelining without hazards is guaranteed:

- a read will not be accepted if the corresponding queue is empty
- whenever a read is accepted, the position of all elements (except bottom) is decremented
- a write will not be accepted if the corresponding queue is full
- whenever a write is accepted, its data is stored on top of the specified queue
- a cell will be output exactly once on exactly the specified output channel
- every cell that leaves the switching element is either empty or it was previously input
- cell sequence is preserved, i.e. any two cells that arrive over one input channel and are targeted to the same output channel will leave the circuit in the order in which they were input
- no central buffer address is dropped from the address management unit

The above requirements represent typical design interface access properties and were characterized by three different formal properties (in the following marked as `nop`, `read`, `write`) which specified the intended design behavior for read, write, and idle accesses. Design behavior was verified by inductively checking that these specifications are met after an initial reset of the design and by showing that, if they hold at an arbitrary point t of time, then they also hold for time $t+1$. Generating the corresponding Bounded Model Checking problems required unrolling of the design over ranges of 4 timesteps (`nop`, `write`), and 6 timesteps (`read`) respectively.

It turned out that for all three properties the width of the data path signals could automatically be reduced by the prototype. As an illustration, RTL block diagrams before and after scaling the design for the `read` property are shown in Figure 10.2.

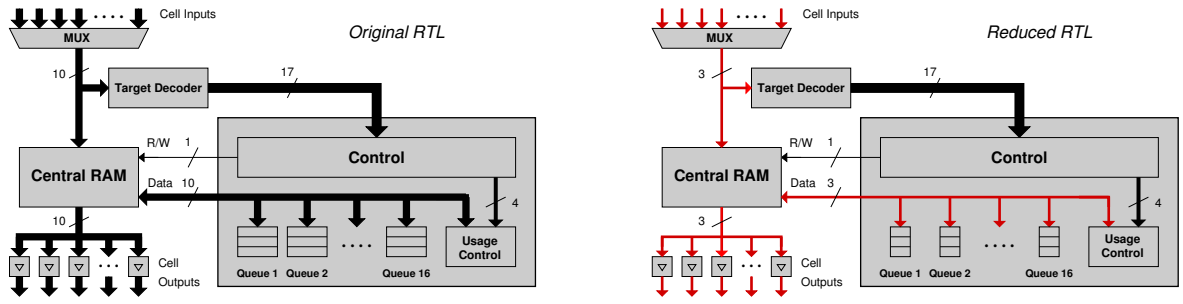


Figure 10.2: Data Path Scaling for the `read`-Property

For all three properties, the verification runtimes on the scaled models of the Q-Unit were compared to those achieved on the original design without preprocessing. All verification test cases were run on an Intel Pentium II PC with a 450 MHz CPU, 128 MB of main memory, and a Linux operating system. The results are given in CPU seconds (minutes respectively) and are shown in Table 10.1.

	Property	Original design	Scaled model
Computation times for pre- and postprocessing	`nop`		2.96 secs
	`read`		6.53 secs
	`write`		3.24 secs
FIFO sizes on RTL	`nop`	160 cells × 10 bit	160 cells × 2 bit
	`read / write`	160 cells × 10 bit	160 cells × 3 bit
Overall number of bits in all signals in cone of influence of property	`nop`	20925	5034 (24.0 %)
	`read`	31452	10592 (33.6 %)
	`write`	14622	5163 (35.3 %)
Overall number of gates in synthesized netlist	`nop`	23801	5661 (27.9 %)
	`read / write`	23801	7929 (33.3 %)
Number of state bits	`nop`	1658	362 (21.8 %)
	`read / write`	1658	524 (31.6 %)
Property checker runtimes	`nop`	23:33 min	37.96 secs (2.7 %)
	`read`	42:23 min	3:27 min (8.1 %)
	`write_fail`	2:08 min	25.66 secs (19.5 %)
	`write_hold`	27:08 min	1:08 min (4.2 %)

Table 10.1: Amounts of Scaling and Verification Runtimes

We encountered a significant reduction in the different sizes of the design models and a tremendous drop in the runtimes of the property checker. Design sizes could be shrunken to approximately 30% of the original sizes, and runtimes dropped from between half and three quarters of an hour to minutes or even seconds.

It turned out that the `write` property did not hold due to a design bug in the Verilog code. A violation of the `write` property occurs if a queue is empty and then receives 3 or more consecutive writes immediately followed by a read. Using the scaled version of the Q-Unit, a counterexample was found in about a quarter of the time needed for the unscaled design (`write_fail`). From the counterexample for the reduced model the prototype recomputed a counterexample for the original design, whereupon the bug was fixed by the designers and the `write` property was again checked on both the scaled and unscaled versions of the corrected design (`write_hold`). Validity could now be proven on the scaled model within fractions of the time needed on the original Q-Unit.

Furthermore, it is especially notable that the computation times which the prototype needed to analyze the design and the properties and to generate the shrunken models, ranging between 3 and 7 seconds, are in effect negligible compared to the runtimes of the property checker. As a conclusion, applying additional pre- and postprocessing truly caused a significant speed-up of design verification.

10.2 Bus Interface Unit

Further evaluations of the prototype implementation of the proposed reduction technique were done at the design center of ARM Limited in Cambridge, UK, one of the industry's leading providers of 16/32-bit embedded RISC microprocessors.

The task present was property checking of a bus interface unit (BIU) which connects an ARM 946 processor core to an AMBA bus (open standard for on-chip bus specification). The design contained several busses, caches and small RAMs. All busses were 40-bits wide. The BIU also contained a write buffer, which was a FIFO into which write data is put if the AMBA bus is busy. The FIFO has 16 entries, with each entry being 40 bits wide (30-bit data + control).

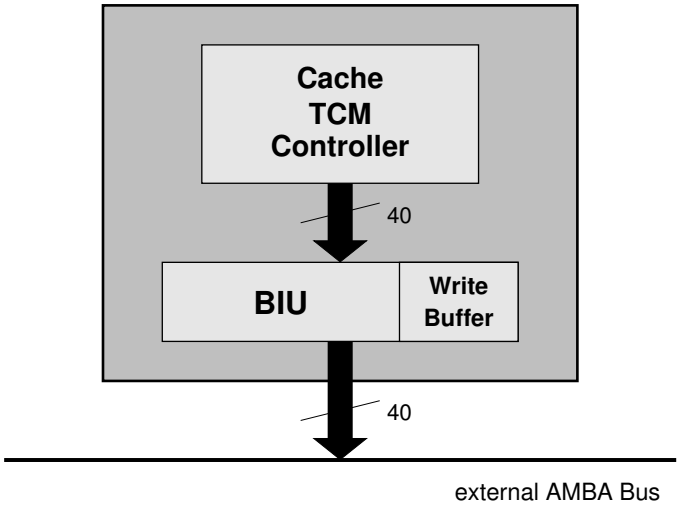


Figure 10.3: Bus Interface Unit

Several standard properties for the bus interface had to be verified, for example that transfers on the bus are AMBA compliant and that read/write requests are correctly handled by the BIU (for instance, no write if the write buffer FIFO is full, no read if the write buffer FIFO is empty). It turned out that the BIU is not safe on its own, i.e. a possible violation of the no-write-if-full property was found when the BIU was checked as a separate module. However, it was unclear whether the BIU maybe still was safe in combination with the control (e.g. in the complete design the control logic should suppress a write-if-full). It was unclear whether the found bug was a true counterexample in the sense that it described a reachable state of the entire design (violations of the property in a non-reachable state could have been neglected).

The problem was that the BIU itself was of a moderate size, just 13.000 gates, and property checking of the isolated BIU block could be done in reasonable time, but the combined BIU plus 946-control and caches was about 68.000 gates.

In a subsequent step, further properties were written which checked for reachability of the found counterexample for the complete bus interface block within a fixed finite window of time (*Bounded* Model Checking), starting from an initial reset state, i.e. property φ_i for example checks if the counterexample can be reached from the initial reset state of the design within i clock cycles. Naturally, the larger the number of clock cycles is, the more grows the size of the bit-level Bounded Model Checking problem (cf. unrolling of design and property, see e.g. Figure 2.3 in Section 2.1). The computational resource limitations were soon reached, i.e. the property checker only got around 10 clock cycles before the proofs became too long.

The verification engineer responsible for checking reachability of the counterexample attempted a scaling of the data path width by manually modifying the HDL source code of the design. According to the design specification, the 40 bits of the bus were conceptually divided into blocks consisting of 2 parity bits, 30 data bits, and 8 control bits, as illustrated in Figure 10.4.

Figure 10.4: Bus Interface – Conceptual Data Flow

The first try consisted of replacing all 30 data bits by one single data bit, thus reducing the overall bus width from 40 down to 11. The effect was that now several properties failed on scaled version of the bus interface which had been successfully verified on the original design, and vice versa. Obviously, the amount of scaling that had been chosen had led to an abstraction which was not one-to-one. In a second trial-and-error attempt, the 30 data bits were replaced by 4 data bits. Yet, again the resulting abstraction was found not to be one-to-one. Considering the effort of manually modifying the whole HDL code (take into account that all signals in the cone of influence of the bus signals are affected by the scaling and had to be modified) it was decided not try further scaling by hand.

During a three-days visit to ARM, the BooSTER tool was used to analyze the bus interface unit and to try to reduce the width of the data path of the design. The bitwise decomposition which was computed by the prototype revealed a granularity of the bus interface signals (cf. Section 6.7) which was the following: the two parity bits and all 8 control bits were recognized to have non-uniform data dependencies and thus to be non-scalable, but for the 30 data bits the data flow was detected to be bitwise, though different for the lower 8 bits and the upper 22 bits. The minimum-width computation yielded a possible reduction of the upper 22 bits down to one bit, but for the lower 8 bits indicated a necessary minimum width which was still 8, as summarized in Figure 10.5.

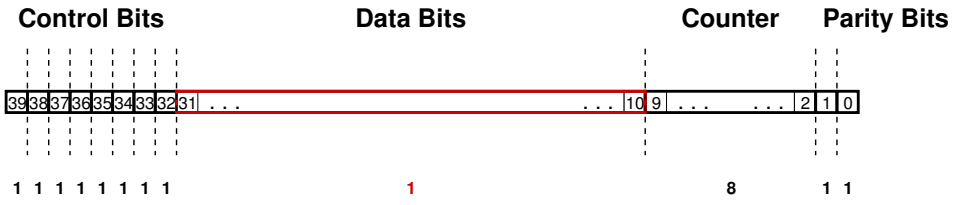


Figure 10.5: Bus Interface – Granularity Analysis of the Data Flow

A close inspection of the HDL code by the chip designer confirmed that the lower 8 bits of the data 30 bits were internally used as some kind of counter. The value of this counter is compared with other signal values at several points on the control path, causing dynamical data

dependencies and disequality constraints for the **BvSAT** problems of the bitwise decomposition. Thus, all 8 counter bits were significant for the considered property and therefore could not be scaled. This had led to the wrong results of the manual attempts of scaling. The correct minimum scaling, as confirmed by the prototype, can be seen in Figure 10.6.

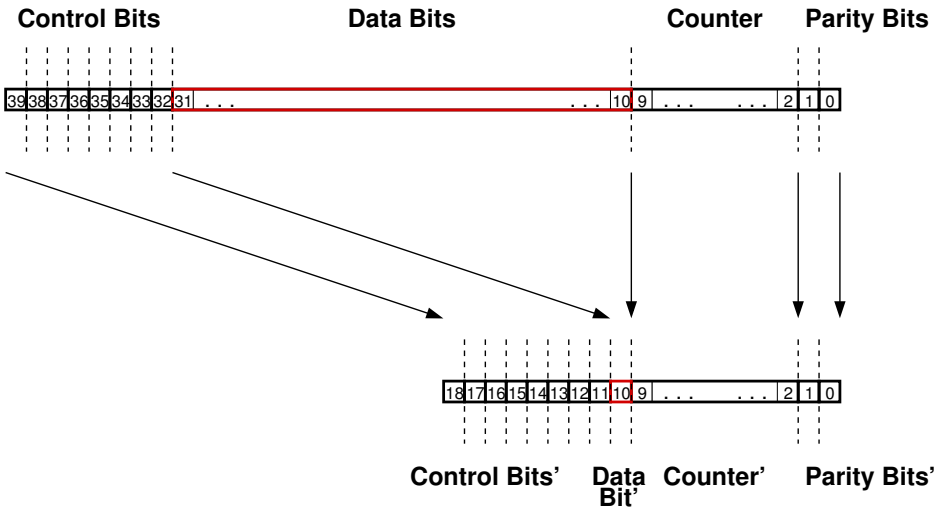


Figure 10.6: Reduced Bus Interface

The result was a possible reduction of the overall bus width from 40 bits down to 19 bits. The scaling is illustrated below.

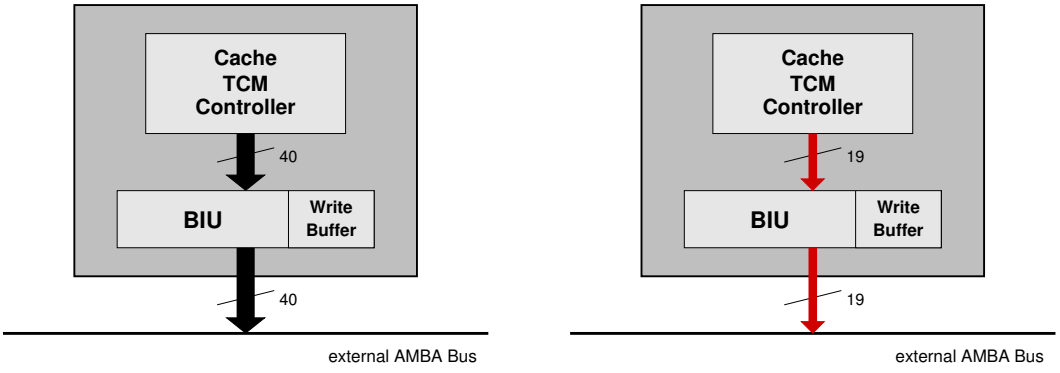


Figure 10.7: Scaling of the Bus Interface

The windows of time for which reachability of the counterexample could be checked when the scaled version of the BIU was used was significantly larger than before. Non-reachability of the counterexample could now be proven for up to approximately 35 time-frames instead of only 10 time-frames, confirming the usefulness of the proposed abstraction technique. Additionally,

the reduction technique was able to improve runtimes of the property checker for several other properties (accurate figures are available only in a confidential internal ARM report). Though here the speed-up was somehow more moderate – runtimes dropping from a few minutes down to under a minute – this was still good news for regression testing, where the aim is to minimize the total running time of all proofs.

Chapter 11

Conclusion and Directions of Future Research

Reducing runtimes and the amount of memory needed in verification computations is one requirement in order to match today's sizes of real world designs in hardware verification. This thesis presents a one-to-one abstraction technique for RTL property checking of digital designs. Given a high-level circuit description and a formal property, the data paths of the design are scaled down by automatically reducing the widths of input, output and internal signals while strictly preserving validity of the property.

11.1 Summary

Constantly growing design sizes of digital circuits require ever improving verification methods. Reduction of data path widths has always been a classical attempt of minimizing state space explosion for formal model checking methods. Many EDA companies today perform such reductions manually to reduce verification runtimes, usually without having the guarantee that the chosen amount of scaling does not falsify verification results.

This thesis proposes a fully automated one-to-one RTL abstraction technique which is used as a pre- and postprocess for property checking of digital circuits. Designs are scaled down by reducing signal widths before property checking while guaranteeing that the property holds for the scaled model if and only if it holds for the original design. If a property does not hold, then counterexamples for the original design are computed in a postprocessing step from counterexamples for the reduced model. Due to providing a one-to-one abstraction, false-negatives cannot occur.

The preprocessing of designs and the postprocessing of counterexamples are strictly separated from the property checking process itself. Thus, the proposed method can easily be integrated into existing verification flows and is independent of the concrete realization of the property checker. It can be combined with a variety of existing verification techniques. The proposed abstraction is particularly well suited to SAT and BDD based hardware verification because the

complexity of those techniques often depends on the number of bit-level variables such provers have to deal with. In classical SAT and BDD based circuit verification such variables are created (at least) for each single bit of each signal of the circuit. In Bounded Model Checking multiple instances of each variable and each signal are created for each step of the considered interval of time such that the effect of scaling is even multiplied. A linear reduction of the width of a data path from n bits down to m bits with $m < n$ effects an exponential reduction of the induced state spaces of the related circuit signals from 2^n down to 2^m. Reduced state spaces result in increased verification performance and allow larger designs and more complex properties to be verified.

The proposed method is based on a one-to-one correspondence which reduces satisfiability problems for bitvector functions of fixed width to satisfiability problems for bitvectors of smaller width. Technically, this is described by instances of the **BvSAT** problem, which is introduced in this thesis and which provides the formal background of the presented framework. Detailed proofs for the correctness of the one-to-one abstraction are given, and it is shown that the correspondence yields minimal reduced models with respect to bitwise bitvector functions.

The reduction technique was tested in several case studies, and experiments on large industrial circuits have demonstrated its applicability and efficiency. The test cases have shown that the reduction technique is able to significantly reduce the runtimes of existing verification tools and to speed up property checking for specific types of digital hardware designs. The approach is well qualified for designs which provide a high degree of uniform data flow and for which the related sets of standard properties, which have to be verified, describe aspects of the data path which also can be characterized by uniform data dependencies. This is typically the case for designs in which data packages are routed from one point to another without extensive modification of the contents of the packages. Standard properties for such designs, for example, check that packages are correctly routed, safely stored, or output in a specific order. These characterizations apply to important design classes such as memories, queues, stacks, bridges, and interface protocols. A further merit of the proposed approach is that, if preprocessing yields that no reduction is possible for a given design and a property, then abstract model and original design are identical. Thus, the verification task itself is not impaired by using the proposed abstraction as a preprocess, the more so as in all case studies pre- and postprocessing runtimes were negligible.

11.2 Ongoing and Future Research

The proposed abstraction technique, as presented in this thesis, offers the basis for several extensions and further improvements. Some promising approaches are outlined in the following sections.

11.2.1 Tweaking the Amount of Scaling

Theorem 4.44 computes the global minimum reduced bitvector width which is necessary and sufficient to guarantee a one-to-one correspondence of satisfiability of the **BvSAT** problems for *all* k-ary bitwise bitvector functions. The only prerequisite is that the data flow which is described

by the **BvSAT** problems is uniform. The amount of reduction which is computed is valid no matter how the specific uniform data dependencies are in detail.

On the one hand, this generality allows a very simple computation of the reduced width (computation of the number of connected components of an undirected graph). On the other hand, often more detailed information on uniform data flow is available which can be further exploited and which can be used for specializations of Theorem 4.41. For example, if an if-then-else expression compares the values of two bitvectors $\boldsymbol{a}_{[n]}, \boldsymbol{b}_{[n]}$, and if there do not exist any further functional dependencies between $\boldsymbol{a}_{[n]}$ and $\boldsymbol{b}_{[n]}$, then an improved amount of scaling can be computed. The reduced width which is computed by the abstraction technique presented in this thesis is conservative in the sense that the resulting abstraction is one-to-one for *any* further functional dependencies between $\boldsymbol{a}_{[n]}$ and $\boldsymbol{b}_{[n]}$.

However, in general such specializations require more complex and costly graph analysis techniques for the disequality graphs. Some of our experiments used two-layered graph data structures and showed promising results.

11.2.2 Improved Scaling of Memory Arrays

Another extension of the proposed reduction technique considers improvements regarding the handling of memory modules in circuit specifications. Up to this point, the frontend which generates the RTL representation of the Bounded Model Checking problem represents a memory module consisting of d cells of n bit width by a single bitvector of width $d \times n$. The computation of the bitwise decomposition causes (at least) a slicing of this bitvector into d chunks, each of width n, corresponding to the d memory cells. The important point is that once the decomposition has been computed, all these cells have the same granularity, which results in the same amount of scaling for each cell, i.e. in the abstract model of the design the $d \times n$ memory module is replaced by a $d \times m$ memory module with $m \leq n$.

The abstraction technique which is proposed in this thesis in general is also capable of reducing not only the size of the individual memory cells, but also to shrink the number of cells itself. Within the bitvector equations, accesses to arrays of memory cells can be described the read and write operators introduced in Definitions 3.15 and 3.16. If the proposed reduction technique detects a possible scaling of the index bitvector, say from a bits to $b \leq a$ bits, then this is an indication that it is sufficient for verifying the property to access only the first 2^b cells of the array instead of all 2^a cells. Thus, the presented abstraction can be enhanced such that $d \times n$ memory modules can be replaced by $e \times m$ memory modules with $e \leq d$ and $m \leq n$. This can drastically multiply the effect of scaling. We are currently investigating such an extension. Several experimental results on real world designs have already confirmed its applicability.

11.2.3 Bitvector Arithmetics

The bitvector decomposition and reduction technique can be improved in order to provide a better handling of arithmetics. Up to this point, data paths which contain arithmetics cannot be scaled at all, because arithmetic operations do not have uniform data dependencies. However, although if-then-else operations do not have uniform data dependencies either, in Chapter 8

we have seen how structural data dependencies and data dependencies caused by if-then-else expressions can be conceptually separated. Additional disequality constraints are added to the **BvSAT** problems, and we have seen that this extension allows for a significantly better amount of scaling.

A similar approach can be taken for arithmetic expressions by excluding them from the decomposition computation in the same fashion as it has been done with if-then-else expressions. The **BvSAT** problem can be further extended by additional arithmetic constraints. Reduction theorems for this type of further enhanced **BvSAT** problems are currently investigated, and first preliminary results are present. An early conclusion is that, if a data path contains only a small amount of arithmetics, say, for example, just one or two adders, then still a good amount of scaling can be achieved.

11.2.4 High-Level Equivalence Checking and High-Level Simulation

A further promising field of application, which is closely related to property checking, is *high-level equivalence checking* of digital designs. Equivalence checking tries to formally verify functional equivalence of two given designs. A typical case is the comparison of a high-level design description (HDL code) and a low-level netlist which, for example, has been generated by a design compiler. The aim then is to verify correctness of the compilation process. As another example, equivalence checking is applied when minor modifications are made to a design, e.g. for logic optimization or layout purposes. Equivalence checking can be used to assure that such modifications do not alter circuit behavior.

High-level equivalence checking can be considered a special case of high-level property checking. Instead of only one design specification, two high-level design specifications are used which describe different implementations of the same circuit. A formal property is generated which checks for functional equivalence. Data path reductions are then computed for the conjunction of both versions of the design with respect to this property. Scaling is then applied to both designs, and the two scaled models are then used to perform the equivalence check. This can be extremely useful already in very early design stages, for example, if a designer just changes some lines of HDL code (e.g. in order to optimize the code or to replace the implementation of a specific module by a different one) and then wants to verify that design functionality is still the same.

It is also conceivable to use automated scaling in *high-level simulation*. Simulation is still an important part of the design flow, and large regression test benches can significantly faster be simulated on smaller versions of a design. Here, the proposed postprocessing of counterexamples provides all means which are necessary to transfer simulation results back to the original design.

Bibliography

[AH97] H. R. Andersen and H. Hulgaard. "Boolean Expression Diagrams". In *Proc. LICS*, pages 88–98, 1997.

[BC94] R. E. Bryant and Y. A. Chen. "Verification of Arithmetic Functions with Binary Moment Diagrams". Technical report, CMU-CS-94-160, 1994.

[BCC+99] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. "Symbolic Model Checking Using SAT Procedures instead of BDDs". In *Proc. DAC*, pages 317–320, 1999.

[BCCZ99] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. "Symbolic Model Checking without BDDs". In *Proc. TACAS*, pages 193–207, 1999.

[BCRZ01] A. Biere, E. M. Clarke, R. Raimi, and Y. Zhu. "Bounded Model Checking Using Satisfiability Solving". *Formal Methods in System Design*, 19(1):7–34, 2001.

[BD02] R. Brinkmann and R. Drechsler. "RTL Data Path Verification using Integer Linear Programming". In *Proc. VLSI Design*, pages 741–746, 2002.

[BDL96] C. W. Barrett, D. L. Dill, and J. R. Levitt. "Validity Checking for Combinations of Theories with Equality". In *Proc. FMCAD*, pages 187–201, 1996.

[BDL98] C. W. Barrett, D. L. Dill, and J. R. Levitt. "A Decision Procedure for Bitvector Arithmetic". In *Proc. DAC*, pages 522–527, 1998.

[BGV99] R. E. Bryant, S. M. German, and M. N. Velev. "Exploiting Positive Equality in a Logic of Equality with Uninterpreted Functions". In *Proc. CAV*, pages 470–482, 1999.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[BP98] N. Bjørner and M. C. Pichora. "Deciding Fixed and Non-fixed Size Bit-vectors". In *Proc. TACAS*, pages 376–392, 1998.

[Bri01] R. Brinkmann. "Using Symmetry for Problem Reduction in Bounded-Model-Checking on the Register-Transfer-Level". In *CP'01 Post-Conference Workshop*, 2001.

[BRTF99] V. Boppana, S. P. Rajan, K. Takayama, and M. Fujita. "Model Checking Based on Sequential ATPG". In *Proc. CAV*, pages 418–430, 1999.

[Bry86] R. E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[Bry92] R. E. Bryant. "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams". *ACM Computing Surveys*, 24(3):293–318, 1992.

[Bry95] R. E. Bryant. "Binary Decision Diagrams and Beyond: Enabling Techniques for Formal Verification". In *Proc. ICCAD*, pages 236–243, 1995.

[BS97] R. J. Bayardo and R. Schrag. "Using CSP Look-Back Techniques to Solve Real-World SAT Instances". In *Proc. AAAI/IAAI*, pages 203–208, 1997.

[BS01] J. Bormann and C. Spallinger. Formal verification for non-formalists. *IT+TI*, January 2001.

[CEJS98] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. "Symmetry Reductions in Model Checking". In *Proc. CAV*, pages 147–158, 1998.

[CFF+01] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi. "Benefits of Bounded Model Checking at an Industrial Setting". In *Proc. CAV'01*, pages 436–453, 2001.

[CFJ93] E. M. Clarke, T. Filkorn, and S. Jha. "Exploiting Symmetry In Temporal Logic Model Checking". In *Proc. CAV*, pages 450–462, 1993.

[CGL92] E. M. Clarke, O. Grumberg, and D. E. Long. "Model Checking and Abstraction". In *Proc. POPL*, pages 342–354, 1992.

[CGLR96] J. Crawford, M. L. Ginsberg, E. Luck, and A. Roy. "Symmetry-Breaking Predicates for Search Problems". In *KR'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, 1996.

[CGPR90] P. Camurati, M. Gilli, P. Prinetto, and M. S. Reorda. "The Use of Model Checking in ATPG for Sequential Circuits". In *Proc. CAV*, pages 86–95, 1990.

[Che91] K. T. Cheng. "An ATPG-Based Approach to Sequential Logic Optimization". In *Proc. ICCAD*, pages 372–375, 1991.

[CKRZ01] M. Ciesielski, P. Kalla, B. Rouzeyre, and Z. Zeng. "Taylor Expansion Diagrams: A new Representation for RTL Verification". In *Proc. HLDVT*, pages 70–75, 2001.

[CMR96] D Cyrluk, M. O. Möller, and H Rueß. "An Efficient Decision Procedure for a Theory of Fixed-Sized Bitvectors with Composition and Extraction". Technical report, Ulmer Informatik-Berichte, Fakultät für Informatik, Universität Ulm, 08 1996.

[CMR97] D. Cyrluk, M. O. Möller, and H Rueß. "An Efficient Decision Procedure for the Theory of Fixed-Sized Bit-Vectors". In *Proc. CAV*, pages 60–71, 1997.

[CZ95] E. M. Clarke and X. Zhao. "Word-Level Symbolic Model Checking - A New Approach for Verifying Arithmetic Circuits". Technical Report 161, CMU-CS, 1995.

[DBR97] R. Drechsler, B. Becker, and S. Ruppertz. "The K*BMD: A Verification Data Structure". *IEEE Design and Test of Computers*, 14(2):51–59, 1997.

[DJ90] N. Dershowitz and J. P. Jouannaud. *"Handbook of Theoretical Computer Science, Formal Models and Semantics"*, chapter "Rewrite Systems", pages 243–320. J.v. Leeuwen, Elsevier, 1990.

[DP60] M. Davis and H. Putnam. "A Computing Procedure for Quantification Theory". *JACM*, 7(3):201–215, 1960.

[Dre00] R. Drechsler. *Formal Verification of Circuits*. Kluwer Academic Publishers, 2000.

[ET99] E. A. Emerson and R. J. Trefler. "From Asymmetry to Full Symmetry: New Techniques for Symmetry Reduction in Model Checking". In *Proc. CHARME*, pages 142–156, 1999.

[FDK98] F. Fallah, S. Devadas, and K. Keutzer. "Functional Vector Generation for HDL Models Using Linear Programming and 3-Satisfiability". In *Proc. DAC*, pages 528–533, 1998.

[GAK99] M. K. Ganai, A. Aziz, and A. Kuehlmann. "Enhancing Simulation with BDDs and ATPG". In *Proc. DAC*, pages 385–390, 1999.

[HC00] C. Y. Huang and K. T. Cheng. "Assertion checking by combined word-level ATPG and modular arithmetic constraint-solving techniques". In *Proc. DAC*, pages 118–123, 2000.

[HD98] S. Höreth and R. Drechsler. "Dynamic Minimization of Word-Level Decision Diagrams". In *Proc. DATE*, pages 23–26, 1998.

[HD99] S. Höreth and R. Drechsler. "Formal Verification of Word-Level Specifications". In *Proc. DATE*, pages 52–58, 1999.

[HIKB96] R. Hojati, A. J. Isles, D. Kirkpatrick, and R. K. Brayton. "Verification Using Uninterpreted Functions and Finite Instantiations". In *Proc. FMCAD*, pages 218–232, 1996.

[ID93] C. N. Ip and D. L. Dill. "Better Verification Through Symmetry". In *Proc. CHDL*, pages 97–112, 1993.

[JD01] P. Johannsen and R. Drechsler. "Formal Verification on the RT-Level – Computing One-To-One Design Abstractions by Signal Width Reduction". In *Proc. VLSI*, pages 127–132, 2001.

[JD02a] P. Johannsen and R. Drechsler. "Speeding Up Verification of RTL Designs by Computing One-To-One Abstractions with Reduced Signal Widths". In *VLSI 2001 Post Conference Book*. Kluwer Academic Publishers, 2002.

[JD02b] P. Johannsen and R. Drechsler. "Utilizing High-Level Information for Formal Hardware Verification". In J. Soldek and J. Pejas, editors, *Advanced Computer Systems*, pages 419–431. Kluwer Academic Publishers, 2002.

[Joh99] P. Johannsen. "On Solving Systems of Bitvector Equations – An Efficient Decision Procedure for a Theory of Fixed-Size Bitvectors with Concatenation, Extraction and Negation". Technical report, Siemens Corp., CT SE 4, 1999.

[Joh01a] P. Johannsen. "BooSTER : Speeding Up RTL Property Checking of Digital Designs by Word-Level Abstraction". In *Proc. CAV*, pages 373–377, 2001.

[Joh01b] P. Johannsen. "Ein Verfahren zur Reduktion von Modellgrößen in Hardware-Verifikationsaufgaben unter Verwendung von Signalbreiten-Skalierungen". *European and US Patent Application*, No. 2001 P 06030, 2001.

[Joh01c] P. Johannsen. "Reducing Bitvector Satisfiability Problems to Scale Down Design Sizes for RTL Property Checking". In *IEEE Proc. HLDVT*, pages 123–128, 2001.

[Lar92] T. Larrabee. "Test Pattern Generation using Boolean Satisfiability". *IEEE Trans. on Computer Aided Design*, 11(1):4–15, 1992.

[LS01] I. Lynce and J. P. M. Silva. "Improving SAT Algorithms by Using Search Pruning Techniques". In *Proc. CP*, page 770, 2001.

[MMM+01] C. F. Madigan, S. Malik, M. W. Moskewicz, L. Zhang, and Y. Zhao. "Chaff: Engineering an Efficient SAT Solver". In *Proc. DAC*, pages 530–535, 2001.

[MR98] M. O. Möller and H. Rueß. "Solving Bit-Vector Equations". In *Proc. FMCAD*, pages 36–48, 1998.

[PRSS99] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. "Deciding Equality Formulas by Small Domains Instantiations". In *Proc. CAV*, pages 455–469, 1999.

[SHS97] S. Scott Hazelhurst and C.-J. H. Seger. Symbolic trajectory evaluation. In Thomas Kropf, editor, *Formal Hardware Verification*, volume 1287 of *Lecture Notes in Computer Science*, pages 3–78. Springer, 1997.

[Sht00] O. Shtrichman. "Tuning SAT Checkers for Bounded Model Checking". In *Proc. CAV*, pages 480–494, 2000.

[Sht01] O. Shtrichman. "Pruning Techniques for the SAT-Based Bounded Model Checking Problem". In *Proc. CHARME*, pages 58–70, 2001.

[Sil95] J. P. M. Silva. *"Search Algorithms for Satisfiability Problems in Combinational Switching Circuits"*. PhD thesis, University of Michigan, 1995.

[SS96] J. P. M. Silva and K. A. Sakallah. "GRASP - a new search algorithm for satisfiability". In *Proc. ICCAD*, pages 220–227, 1996.

[SS97] J. P. M. Silva and K. A. Sakallah. "Robust Search Algorithms for Test Pattern Generation". In *Proc. FTCS*, pages 152–161, 1997.

[SS99] J. P. M. Silva and K. A. Sakallah. "GRASP: A Search Algorithm for Propositional Satisfiability". *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[SS00] J. P. M. Silva and K. A. Sakallah. "Boolean satisfiability in electronic design automation". In *Proc. DAC*, pages 675–680, 2000.

[Stå95] G. Stålmarck. "Method and Apparatus for Checking Propositional Logic Theorems in System Analysis". *European Patent*, No. 0403 454, 1995.

[VB98] M. N. Velev and R. E. Bryant. "Bit-Level Abstraction in the Verification of Pipelined Microprocessors by Correspondence Checking". In *Proc. FMCAD*, pages 18–35, 1998.

[VB99] M. N. Velev and R. E. Bryant. "Exploiting Positive Equality and Partial Non-Consistency in the Formal Verification of Pipelined Microprocessors". In *Proc. DAC*, pages 397–401, 1999.

[WAH01] P. F. Williams, H. R. Andersen, and H. Hulgaard. "Satisfiability Checking Using Boolean Expression Diagrams". In *Proc. TACAS*, pages 39–51, 2001.

[Zha97] H. Zhang. "SATO: An Efficient Propositional Prover". In *Proc. CADE*, pages 272–275, 1997.

[ZKC01] Z. Zeng, P. Kalla, and M. Ciesielski. "LPSAT: A Unified Approach to RTL Satisfiability". In *Proc. DATE*, pages 398–402, 2001.

[ZKCR01] Z. Zeng, P. Kalla, M. Ciesielski, and B. Rouzeyre. "Functional Test Generation using Constraint Logic Programming". In *Proc. VLSI-SOC*, 2001.

List of Figures

List of Tables

List of Algorithms